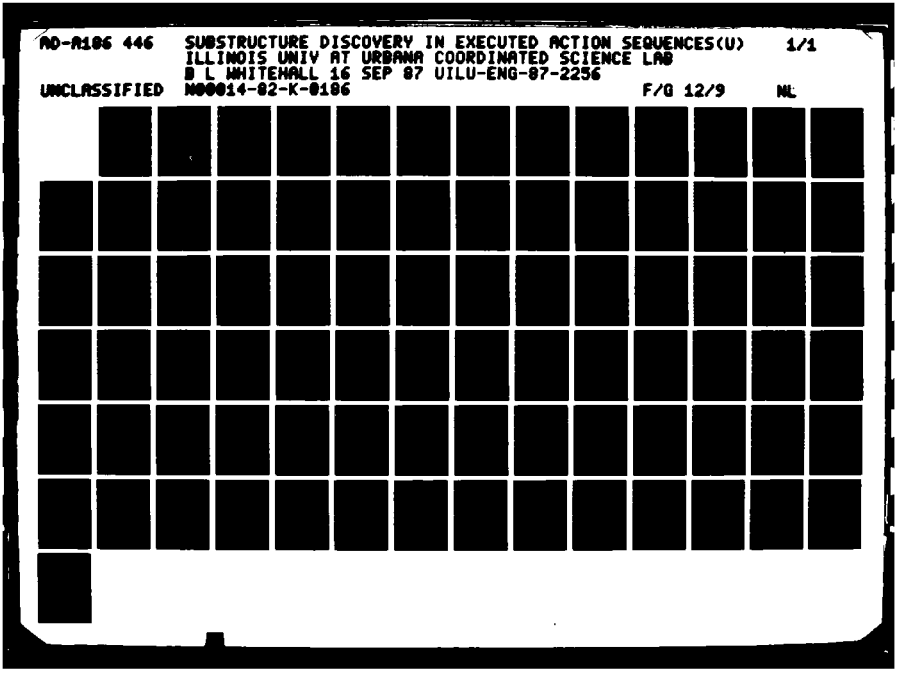
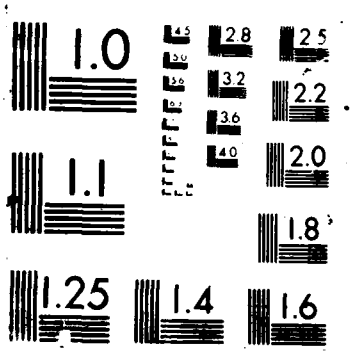

AD-A186 446 SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES(U) 1/1
ILLINOIS UNIV AT URBANA COORDINATED SCIENCE LAB
B L WHITEHALL 16 SEP 87 UILU-ENG-87-2256
UNCLASSIFIED N00014-82-K-0186 F/G 12/9 NL


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

September 1987 UILU-ENG-87-2256

(2)

COORDINATED SCIENCE LABORATORY
College of Engineering

AD-A186 446

SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES

Bradley Lane Whitehall

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) [redacted]	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois	6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION National Science Foundation, Office of Naval Research, Defense Adv. Research Projects Agency
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801		7b. ADDRESS (City, State, and ZIP Code) NSF: 1800 G St., N.W. Washington, D.C. 20550 ONR: 800 N. Quincy St. Arlington, VA 22217 (over)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION NSF, ONR, and DARPA	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER NSF: IST-85-11170 · ONR: N00014-82-K-0186 DARPA: N00014-85-K-0078
8c. ADDRESS (City, State, and ZIP Code) See block 7b.		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES

12. PERSONAL AUTHOR(S)
Whitehall, Bradley Lane

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	87, 09- 16	73

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
			Substructure Discovery, Macro operators, Background knowledge, similarity-differenced based learning, conceptual clustering

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis describes a system, PLAND, for discovering substructures in observered action sequences. The goal of this thesis is to show how a system can learn useful macro operators by observing a task being performed. An intelligent robot using this system can learn how to perform new tasks by watching tasks being performed by someone else, even if the robot does not possess a complete understanding of the actions being observed. There may be more than one hypothesis of how an action (or macro operator) contributes to the completion of a task and PLAND allows for these various meanings to be pursued simultaneously.

Macro operators are discovered within a specific context that provides the types of generalizations allowed in the discovery process and uses the previously proposed macro operators to build new ones. Background knowledge is used to determine which generalizations are appropriate and to control search. Although the system can discover simple syntactic structures (regular grammars) without background knowledge, more meaningful and useful structures are discovered when background knowledge is incorporated into the process.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

7b. ADDRESS (continued)

DARPA: 1400 Wilson Boulevard
Arlington, VA
22209-2308

19. ABSTRACT (continued)

The foundations of PLAND are in similarity-differenced based learning systems that perform conceptual clustering. however this system incorporates more background knowledge than previous systems to improve the learning capabilities of the system. Sample tasks of discovering macros for moving boxes between adjacent rooms and understanding a student's daily routine are presented to demonstrate the capabilities of the system.

September 1987 UILU-ENG-87-2256

COORDINATED SCIENCE LABORATORY
College of Engineering

SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES

Bradley Lane Whitehall

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES

BY

BRADLEY LANE WHITEHALL

B.S., Northern Illinois University, 1983

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1987

Urbana, Illinois

SUBSTRUCTURE DISCOVERY IN EXECUTED ACTION SEQUENCES

Bradley Lane Whitehall
Department of Computer Science
University of Illinois at Urbana-Champaign, 1987
Thesis advisor: Robert E. Stepp

This thesis describes a system, PLAND, for discovering substructures in observered action sequences. The goal of this thesis is to show how a system can learn useful macro operators by observing a task being performed. An intelligent robot using this system can learn how to perform new tasks by watching tasks being performed by someone else, even if the robot does not possess a complete understanding of the actions being observed. There may be more than one hypothesis of how an action (or macro operator) contributes to the completion of a task and PLAND allows for these various meanings to be pursued simultaneously.

Macro operators are discovered within a specific context that provides the types of generalizations allowed in the discovery process and uses the previously proposed macro operators to build new ones. Background knowledge is used to determine which generalizations are appropriate and to control search. Although the system can discover simple syntactic structures (regular grammars) without background knowledge, more meaningful and useful structures are discovered when background knowledge is incorporated into the process.

The foundations of PLAND are in similarity-differenced based learning systems that perform conceptual clustering, however this system incorporates more background knowledge than previous systems to improve the learning capabilities of the system. Sample tasks of discovering macros for moving boxes between adjacent rooms and understanding a student's daily routine are presented to demonstrate the capabilities of the system.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor, Professor Robert E. Stepp, for his insight, guidance, and support. His ability to recognize the important issues in problems has kept this research on the right path. He has taken the work out of my graduate studies and replaced it with highly educational and personally rewarding opportunities.

I would also like to thank Larry Holder, Bob Reinke, and Bharat Rao for their ideas and comments on substructure discovery. Thanks to Ray Mooney, my officemate, for his tolerance and fun attitude that made the office an enjoyable place to work. Also, thanks are due to the other members of the AI group at the Coordinated Science Lab: Jude Shavlik, Steve Chien, Shankar Rajamoney, and Scott Bennett for helping to provide a stimulating environment. Thanks to Brian Mogensen for taking time to help me get started on the TI Explorer when he had better things to do.

A special thanks goes to my mother, father, and sister who have supported all I have done even when they did not understand what I was doing.

This research was conducted at the Coordinated Science Laboratory of the University of Illinois at Urbana-Champaign. Support was provided by the National Science Foundation under grant NSF IST-85-11170, the Office of Naval Research under grant N00014-82-K-0186, by the Defense Advanced Research Projects Agency under grant N00014-85-K-0078, and by a gift from Texas Instruments Incorporated.

TABLE OF CONTENTS

CHAPTER

CHAPTER

CHAPTER 1

INTRODUCTION

Learning through discovery is important in many areas of research. For centuries man has labored to learn more about the environment in which he survives. The ability to reduce the complexity of phenomenon has enabled the understanding of complex entities ranging from the atom to deep space. Much of science has been concerned with breaking events into smaller pieces in order to study those parts. The smaller components are simpler to study and facilitate a better understanding of the whole. Other aspects of science have worked on explaining objects by grouping elementary components into structures. There are fewer structures than elementary components in the new system to consider. Viewing the structure as macroscopic parts allows the more important relationships between the parts to become apparent. This thesis describes a system, PLAND, that discovers plan macros by an agglomerative clustering of plan (action) subsequences.

PLAND (PLAN Discovery) is a system that learns useful operations and groups important sequences of actions into bundles by observing someone else performing a task. The goal of this thesis is to show that a system can learn useful sequences of actions by observing the execution of a task without understanding the justification for all the steps performed. This is an old idea and is one of the foundations of apprenticeships: an apprentice is able to learn from the master craftsman by observing his skill in action. The algorithms presented allow PLAND to learn in a similar manner.

If a system could understand a set of operations by dealing individually with the primitive actions at a single level of abstraction, then the task of substructure discovery would be simple. But in solving a problem there is a hierarchy of goals that have been met. The observer may know

the highest level goal but not the intermediate goals. Learning only the useful operations at the lowest level is not enough. The system must also build up the goal structure to aid in the understanding of what has been observed.

The system collects into chunks groups of actions that are observed occurring sequentially and labels these chunks macro operators (macrops) [Fikes72]. Macro operators can be used by the planning system to achieve greater efficiency the next time it has to perform a similar task. There are three types of macro operators that the system may discover: loops, conditionals, and sequences. A loop construct consists of a body of actions that are executed repeatedly until some condition is obtained. Conditionals are structures that allow choice points in the path of execution of the macrop Sequences are simple blocks of actions that appear in many places of the observed action trace. These constructs allow the system to define macrops with the expressiveness of regular grammars.

Possessing only knowledge of these three types of macrops, the system is able to find interesting macrops in very simple domains, but more power is needed to find macro operators in complex domains. Power is obtained through the use of background knowledge specific to the domain in which the task is performed. Background knowledge is used in a variety of ways by the PLAND system, ranging from guiding the execution of the algorithm (meta-knowledge) to determining if specific blocks of actions should be allowed as macrop definitions. PLAND combines similarity-difference based learning (SDBL) algorithms [Hayes-Roth78, Hoff83, Stepp84, Vere78] with knowledge intensive routines [DeJong86, Mitchell86] to discover interesting macrops.

An example will give a better perspective of what PLAND accomplishes. PLAND is built with the objective of helping an intelligent robot learn about the world in which it operates. Assume a robot has lived in the city since being built and suddenly the owner of the robot moves to the suburbs. The owner tells the robot to keep the yard looking nice and points out that the neighbor has a well groomed yard. At this point the robot is befuddled. It cannot use its SDBL algorithms to learn how to clean up the yard as only one example has been presented. It cannot use

explanation based learning (EBL) algorithms — it possesses no domain knowledge about lawns. The robot is intelligent enough to wait and see how the neighbor keeps that yard so nice. The day the neighbor is mowing his yard, the robot is there observing him. The robot records every action that the person makes, even though it does not understand the reasons behind the actions. The robot observes the following actions:

(1) The neighbor walks around the yard stopping to pick up a soda pop can, then a toy, and then a piece of paper.

(2) The person starts the lawn mower.

(3) The mower is pushed back and forth across the yard, changing the path of the mower by its width each time.

(4) When the yard has been traversed with the mower the person shuts off the mower and goes into the house.

Reviewing the sequence of actions just performed the robot is able to figure out two macrops. First, a looping construct is discovered for pushing the mower. The robot generalizes that the exact number of paths made with the mower is irrelevant. With the mowing macrop the robot realizes that the mower cannot be pushed over items that may be cut. It reformulates the actions it has observed to consider all breakable items as equivalent. Now the robot can discover another looping construct that contains a conditional action. The loop is for walking around the yard and picking up garbage, where picking up garbage is a conditional action depending on identifying small breakable things found in yards. The number of breakable items in the yard is irrelevant to the discovery of the loop.

The role of background knowledge is important in this scenario. Knowledge about various artifacts allows the system to generalize over all items that are breakable and discover the conditional plan for picking up items in the yard. Since the robot has macrops for picking up junk

in the yard and pushing the mower, it may mow the yard without having to plan every action at the lowest level and without having a complete (deep) theory of yards.

This thesis describes how the PLAND system can learn by watching the performance of tasks as in the scenario above. Before explaining exactly how the system works, some background is presented in the next chapter. After discussing related work in Chapter 3, Chapter 4 gives some basic definitions for substructure discovery. After that the system is described in Chapter 5 and Chapter 6, followed by examples run on the system (Chapter 7), future work (Chapter 8), and conclusions (Chapter 9). The key points of the thesis are that learning substructures is an important type of discovery and that substructure discovery can take place in knowledge poor domains but better structures can be composed when knowledge is used in conjunction with the discovery process.

CHAPTER 2

BACKGROUND AND MOTIVATIONS

This chapter presents the background information that is needed to understand the remainder of the thesis. Along with the background, some motivations for building a system like PLAND are discussed.

2.1. Substructure and Discovery

The problem studied in this thesis is discovering the structure of plans observed in the world. Specifically, an intelligent robot using this technique can learn from the actions being performed around it, even when the hierarchical structure that dictates the action sequence performed for the task is unknown. Research on PLAND addresses the issue of discovering macro operators (macrops) from a set of observed plans by using *conceptual clustering* techniques. The mechanized discovery of macrops is desirable for many reasons. First, reducing the complexity of an observed event facilitates the understanding of the event as a whole. Second, when an entity must plan how to perform a task, it is easier to plan at a higher level ignoring unimportant details by using macrops. Discovered macrops facilitate planning in a top down fashion. Third, using macrops allows a planner to handle longer plans (more primitive actions) than would otherwise be computationally possible [Fikes72].

Macro operators are a type of substructure. A substructure can be viewed as a collection of nodes and relations between those nodes. When an observed event contains a large number of nodes it is beneficial to reduce the inherent complexity by grouping sets of nodes together and considering them as a single unit, especially in visual perception [Treisman82, Wolff76, Zahn71]. Compaction is also important when planning for the execution of a task. An executed plan is just a

sequence of primitive action steps. Groups of action sequences may be performed many times in an executed plan, and are thus more efficient when considered as a unit. Macro operators (macrops) are these groups of primitive action steps that may be reasoned about as a complete unit. The *cognitive savings* associated with one or more macrops is a numerical measure of the amount of mental effort one gains by using the macrop(s) instead of the primitive actions.

The PLAND system discovers the structure of a plan by forming into macrops logical groupings of actions that perform a definable unit of work. Structure in this case is the relationship between actions used to accomplish the plan. The system learns by chunking [Laird84] actions into macrops. The learning occurs at the knowledge level [Dietterich86b] because the macrops constructed are new forms that the system can use in reasoning. Before the macrops are formed the system has no mechanism for reasoning about the group of actions in the macrop as a complete unit. PLAND discovers three types of macro operators: loops, conditionals, and sequences. These are discussed in detail in Chapter 6.

Learning by observation or discovery is one of the six types of learning identified by Michalski [Michalski86]. Michalski also defines two forms of learning by observation, active and passive. In active observation the system possesses the ability to permute or directly explore the environment; experimentation is possible. In passive observation the system does not have the power to change events; experimentation is not possible. Both types of discovery are important and useful in particular situations. It would be difficult to learn chemistry without performing experiments, but much has been learned about astrophysics without experimentation. This research looks at the area of passive observation for discovering substructures. Substructure discovery is a logical extension to *conceptual clustering* programs, such as the CLUSTER family [Michalski83b, Mogensen87, Stepp84]. The CLUSTER programs group items based upon descriptions given about the input events. Intelligent clustering systems must be able to devise relevant attributes upon which to cluster objects to find clusterings that address the goals of the given task. Substructure discovery is concerned with this problem. Generating substructures requires *part-to-whole*

generalization [Dietterich86a]. In other words, from the pieces the system sees, it determines how the complete event is best described using generalizations of the fragments.

2.2. Background Knowledge

PLAND uses background knowledge to help guide the search for macro operators. Psychological research has shown that analysis of similarities and differences alone does not account for the categories formed when classifying events [Wattenmaker87]. A system needs some notion of context for the classifications being made [Tversky77]. As described in section 5.3, PLAND builds *contexts* for various levels of generalization it uses. The background knowledge indicates which context should be used on a specific pass of the algorithm. Contexts allow the system to alter its views of the input actions. In this way the system can hold more than one possible interpretation for the action steps at any point in time.

The knowledge used depends upon the actual task being performed. If very little about the task is known, then little direction from the background knowledge can be given. In this case the system discovers a regular grammar to describe the input. When the system has much knowledge about the domain, that knowledge can be incorporated to help guide the search for new macrops. Of course, if the system had complete knowledge of the domain there would be no reason to perform the discovery process.

2.3. Additional Motivations

Discovering substructures is an interesting problem partly because it is so closely related to *Gestalt* psychology. Conceptual clustering is based on the premise of dividing objects into conceptually coherent groups. But building substructures allows a system to develop clusterings with a more Gestalt sense. When a system is creating substructures it is defining the "attributes" that make occurrences of the substructures equivalent. If one set of substructures does not adequately describe the input, then another set may be discovered. This level of control is important because it allows the system to find different classifications of the same input events for

very different purposes. The flexibility offered by such a system that interprets its input greatly surpasses the best system which can use only the initially given attributes.

The key to discovering substructures in this manner is background knowledge. To demonstrate this Bongard [Bongard67] presents many sets of drawings and challenges a system to describe the conceptual difference between the two classes in each set. People can perform this task quite well. Before a system could begin to describe the differences between the classes it would have to be able to describe the components of the drawings at a conceptual or Gestalt level. Substructure discovery seems to be the logical approach to solving this problem. The basic knowledge about drawings is limited (lines, circles, above, below, big, small, etc) but the number of combinations explodes when searching for an answer blindly. However, a system that uses knowledge to discover segments within the drawings and then uses constraints implied by those structures might be able to solve the problem. This is an interesting problem, and the research on PLAND and substructure discovery is a very small step in the direction of a solution.

CHAPTER 3

RELATED WORK

The ideas incorporated into PLAND are derived from a variety of other systems. These systems and their relations to PLAND are discussed in this chapter. Some of these systems are related to PLAND by domain, some by virtue of being discovery systems, and some by conceptual parentage. Many systems will be referenced in this chapter and all will be contrasted against PLAND, but not with each other. The CLUSTER family of programs, only abstractly related to PLAND, are covered first. Next, the relation between PLAND and other discovery systems such as GLAUBER, STAHL, AM, and EURISKO will be explained. Then, two programs with special relationships to PLAND are presented: NODDY and SPARC/G. NODDY, like PLAND, learns macrop operators for plans but NODDY learns from teacher supplied examples. SPARC/G is a general qualitative prediction program with much in common with PLAND at the functional level. Next, a few systems that work on building finite state machines or regular grammars from input/output traces are discussed, since PLAND performs this task when not working with background knowledge. Finally, another substructure discovery program, SUBDUE, is discussed.

3.1. CLUSTER Family

The most widely known conceptual clustering programs are the members of the CLUSTER family. They include CLUSTER/2 [Stepp83, Michalski83b], CLUSTER/S [Stepp84], and CLUSTER/CA [Mogensen87]. Conceptual clustering is a form of discovery or learning by observation where a group of items (examples) are classified by attributes and structural relations of the individual items. The clusterings are based on the conceptual cohesiveness [Michalski83b] of

the items within and between the clusters. The algorithms try to obtain high intra-class cohesiveness and low inter-class cohesiveness. CLUSTER/2 works only with attribute-based descriptions, using attributes of the whole object such as color, size, shape, and texture. CLUSTER/S is an extension of CLUSTER/2 which builds clusterings based on attributes and structural components. Structural components allow one to express properties of the subparts of an item and relationships between those parts in the form of N-ary predicates. Examples are on-top(lintel-1,support-2) and component-of(wheel,car). CLUSTER/CA uses a goal dependency network (GDN) [Stepp86] to direct the algorithm to develop better clusters where "better" depends upon the domain and purpose of the clustering. The knowledge stored in the goal dependency network enables CLUSTER/CA to build clusters that more closely fit the circumstances of a particular user's situation.

Two important aspects of the the PLAND system were derived from the CLUSTER systems. The first is the importance of deriving attributes during clustering. Experiments with CLUSTER/S showed that the best attributes for clustering are not always those that are given initially. The system uses inference over background knowledge to generate new attributes for an event using the process known as *constructive induction* [Dietterich83]. In PLAND a process like constructive induction is applied to discover structural patterns of nodes that can subsequently be treated as attributes in the overall structure. The discovered structures are built up from elementary pieces (nodes and relations) by performing *part-to-whole* generalization. The lowest level elementary pieces are the action steps in an event that is an observed execution trace and the glue that connects the actions is the "follows" relation. Discovering internal structure in events is a logical extension to the inductive techniques used in the CLUSTER family of programs.

The second important trait PLAND received from the CLUSTER systems is the notion of using background knowledge to guide the discovery process. CLUSTER/CA uses a GDN to help guide the search for useful clusterings, but compared with PLAND the knowledge takes a more passive role. PLAND uses domain knowledge in an active way, using it to determine levels of

generalization in addition to guiding search. As described in section 5.4, PLAND uses background knowledge at many different levels.

There are some obvious differences between the PLAND system and the CLUSTER programs. The most important difference is that the CLUSTER programs are given distinct items which are to be classified. The PLAND system takes in a trace of an action sequence and does not attempt to group the individual actions into classes. Rather, it connects these actions so that the instances within the formed macrops are similar. Thus the grouping does not occur based upon attributes assigned externally to the items, but on the structure built by the system. Another difference between these systems becomes obvious when looking at the method in which the classes are formed[1]. When building a hierarchical clustering, CLUSTER starts with the most general class and then decomposes it when going down the hierarchy. PLAND on the other hand starts at the bottom, with individual actions, and composes them into macrops. The macrops can be built recursively using other macrops. The structure for the whole input sequence is discovered from the bottom up.

3.2. Scientific Discovery Systems

PLAND has many features in common with other discovery programs. GLAUBER and STAHL [Langley86] are the specific programs discussed in this section. They are members of the BACON family of scientific discovery systems [Langley81]. GLAUBER formulates qualitative empirical laws; the goal of the system is to transform a set of input facts into a set of laws of the same form as the facts but with specific substances being generalized to abstract classes. Figure 3.1 presents the form of the facts given to GLAUBER and the rules discovered [Langley87]. GLAUBER consists of two main operators, FORM-CLASS and DETERMINE-QUANTIFIER. FORM-CLASS takes the input facts and generalizes them by replacing the most common argument in all

[1] Here macrops are defined to be classes of instructions.

Input fact:
(reacts inputs {HCl NaOH} outputs {NaCl})

Output rule:
$\forall$ alkalis $\forall$ acid $\exists$ salt (reacts inputs {acid alkali} outputs {salt})

Figure 3.1: Sample Input and Output from GLAUBER

predicates by a class name. DETERMINE-QUANTIFIER decides if the new class discovered by FORM-CLASS should be universally or existentially quantified.

STAHL is like GLAUBER in that it accepts qualitative statements as input and outputs qualitative statements, but STAHL builds an internal representation for the structures of the substances used in the statements. These structures can be viewed as explanations of the facts in the statements.

PLAND has more in common with these systems than simply being a discovery program. GLAUBER performs its generalizations in multiple levels which can be considered equivalent to PLAND's contexts[2]. These levels are built by allowing FORM-CLASS and DETERMINE-QUANTIFIER to work on the results of a previous level. Like PLAND, GLAUBER works in a bottom up fashion, building the hierarchy by grouping items. Langley et al. [Langley87] point out that there are two different types of aggregation performed by discovery systems. In CLUSTER and GLAUBER, the systems must decide which instances should be grouped together to form classes. In the aggregation performed by PLAND, the system must determine which parts of the input should be grouped to build higher level objects. The difference is subtle but important.

STAHL discovers the components of nonelemental substances involved in chemical reactions. STAHL tries to determine the structure of compound substances by proposing a structure that will

[2] See section 5.3 for details on contexts.

work in all the known reactions. This system proposes structures that meet the constraints of known reactions. As other chemical reactions are presented to the system (working in an incremental fashion) these structures may have to be revised. This differs from the PLAND system where the structures are built up from what is observed and no formula is available to check the correctness of the substructure. In PLAND, the complexity of nesting macrop operators is introduced by the system, not by constraints that are given.

3.3. AM and EURISKO

AM and EURISKO are two of the most important discovery programs written to date [Lenat84]. Starting with a set of mathematical knowledge, AM uses a best-first search strategy based on the interestingness of a concept to discover other interesting concepts. The system uses a frame based representation for the concepts discovered and an agenda system to decide which concepts to pursue in the search. EURISKO builds upon AM and allows the system to modify the heuristics used in the search as well as build new concepts. At a superficial level there are a couple of similarities between PLAND and AM. Like AM, PLAND uses an agenda control system to help control the search for macrops (concepts). Both systems use a metric to determine which agenda items to perform, AM uses interestingness and PLAND uses the sum of the cognitive savings for the macrops defined in the agenda.

These systems share more then surface features when the use of background knowledge is considered. Knowledge allows AM to permute its concepts in meaningful ways to develop interesting concepts. The heuristics are knowledge driven to efficiently guide the system in the best directions. Likewise, PLAND is very dependent upon knowledge to direct its search. Without the use of domain knowledge PLAND would still discover the same macrops[3] but not as quickly. Both systems embody the notion that much knowledge is required to discover interesting concepts.

---

[3] At the level of the actions given, the system without background knowledge could not generalize the actions to produce new contexts.

EURISKO extended AM by allowing heuristics to modify other heuristics as well as concepts. Although PLAND does not directly modify its heuristics, it does allow the background knowledge to determine where it should search for macrops. If the system is pursuing more than one goal, the knowledge may indicate that different heuristics are applicable in the different situations. As the system works on more general levels of the action hierarchy the heuristics may also change. Although PLAND does not currently modify its knowledge, other than to indicate a different working level, there are no reasons why future versions of the system could not modify the knowledge base during processing.

3.4. NODDY

Andreae's NODDY system [Andreae85] has a special relation to the PLAND system. The connection between these systems lies in the notion that discovery may occur through planned iterative discrimination. This is the type of relationship between INDUCE [Hoff83] and CLUSTER/S. NODDY learns generalized procedures from positive examples presented by a teacher. The system is able to generate negative examples from the positive examples given. Thus NODDY learns with the aid of an instructor (INDUCE does likewise) where PLAND learns by observation and must devise its own examples (CLUSTER determines which events are "positive" seeds and "negative" seeds). CLUSTER can conceptually be thought of as selecting initial points (seeds) to represent classes and then using INDUCE to determine descriptions for the classes by extending the points against each other [Stepp84]. At a very abstract level, one could imagine discovering macrops within a trace by breaking it into chunks and letting NODDY process these chunks to build procedures. The problems with such an approach are enormous and the procedure would be inefficient.

Although NODDY and PLAND both produce generalized procedures from sequences, the knowledge used to build those procedures is very different. NODDY performs the minimum generalization possible to cover new examples. Following in the paradigm of Winston's ARCH

system [Winston75], the examples must be presented in a pedagogical manner. PLAND does not perform minimum generalization because an inductive inference is made just to propose a macrop structure. When specific examples are given to a system, it may focus on components that are different from previously encountered events. This allows more specific changes to be incorporated into the built structure. When discovering structures, there does not exist a "standard" to compare with proposed structures. NODDY does not require or use much domain knowledge. Since it is handed examples, the system only needs knowledge on the basic actions and the procedure constructs allowed. PLAND requires much knowledge to help it prune the search space to a manageable size.

One clear example of how these systems differ is in the creation of loops. NODDY does not explicitly produce a looping construct. Loops fall out as a byproduct of the minimal generalization that takes place when an example with a loop is presented. PLAND on the other hand searches intensely for loops because they are a conceptually strong structure for actions.

PLAND and NODDY perform their tasks in the same domain, procedure actions. NODDY works with examples that are known to be correct. Therefore the system can learn some of the conditions for the conditionals that are produced. PLAND works with an unstructured observation. It must build structure on top of the actions in order to facilitate the understanding of the task being performed.

3.5. SPARC/G

SPARC/G is a general purpose prediction program [Michalski87]. Given events and attributes for those events the system determines which attributes are important in correctly predicting what might happen next. Of course the prediction can never be certain, but SPARC/G attempts to constrain the number of possibilities to as small a set as possible. Figure 3.2 illustrates a typical input example to SPARC/G. Given such an example, the system would predict that the next item would contain four nodes. In order to make such predictions the system must model what it has

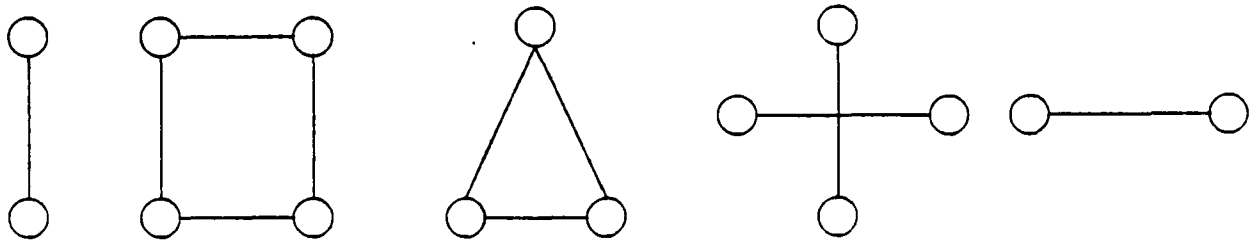


Figure 3.2: SPARC/G Input

observed thus far. This requires part-to-whole generalization, similar to that performed by PLAND. SPARC/G learns what periods exist in the given example in order to predict future events. There are three types of periods defined by the system:

(1) Periodic conjunctive model. This model specifies that the period is a conjunction of phases where each phase is a predicate calculus formula[4]. A period of a blue object followed by a red object is represented as

Period (color(blue), color(red)).

(2) Look back decomposition model. This model requires that the next object type depend upon previously encountered events. The period is in the form of if-then rules. The left hand sides of the rules refer to the previously seen events. Subscripts are used to indicate how far back in the sequence the predicate applies with zero being the next event to be encountered. The rule represented by

$$color_2(red) => color_0(green)$$

means that if the object two places back is red then the next event will be green.

[4] Actually they are VL_1 expressions [Michalski75].

(3) Disjunctive Normal Form. This form allows sequences to be described in a hierarchical fashion as a sequence of subsequences. For example the sequence

$$S = < 3, 4, 4, 5, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7, 7 >$$

can be described by

$$S = <(3,1), (4,2), (5,3), (6,4), (7,5)>$$

that is found by computing the difference between items and recognizing that the next item in the sequence occurs one more time than the previous item.

The periods discovered by SPARC/G and the loop constructs discovered by PLAND are conceptually very close, though the actual representations are very different. The phase of a period is equivalent to the body of a loop. The SPARC/G system uses difference rules to compute the differences between values that it sees. These are analogous to the fuzzy matching rules that PLAND uses to generalize the actions. SPARC/G can find patterns that resemble conditionals but the segment conditions must be provided by the user, where PLAND discovers the break conditions. SPARC/G can easily represent a sequence of increasing items such as

$$<1, 2, 3, 4, 5, 6, 7, 8>$$

where PLAND has no method of representing such patterns.

SPARC/G uses some background knowledge in its processing. The typical example is the user provides the number of phases to expect in a period. SPARC/G is not constrained to work in a single domain but that also means it is not able to take advantage of certain constraints found in the action sequence domain.

PLAND can solve problems that SPARC/G cannot handle. A looping structure with a variable number of iterations of the loop body is discoverable by PLAND. But SPARC/G requires a fixed number of iterations (or a definite pattern as defined above) for each occurrence of the loop in order to discover it. This greatly constrains the type of structure found by SPARC/G and patterns in the action-sequence world rarely occur in this rigid format, even when a clear pattern exists.

3.6. Grammatical Inference

When PLAND does not use background knowledge it finds macrops that are equivalent to finite state machines for chunks of the input. The input example is equivalent to a string where the actions are letters of the string. The set of macrops discovered for an example can be thought of as a grammar to define the input example and other "similar" strings. The first part of this section describes some previous work in grammatical inference and the tenuous connections between this research and PLAND. The second part of this section discusses the SNPR system of Wolff and the similarities PLAND has with it.

3.6.1. Learning Grammars from Examples

Much research has occurred in the area of learning grammars from examples. The details of this work will not be presented due to the volume of material and the slight relevance to the PLAND system. This section will present a quick overview of this research and show where PLAND fits in this scheme. Much of the research in this area has dealt with phrase structure grammars, not just regular grammars.

There are three methods that have been used to learn context free grammars: enumerative, constructive, and refinement [Cohen82]. Enumerative methods generate grammars one by one and test each to determine how well it describes the examples. Pao and Carr [Pao78] developed a system that constructs a lattice of the possible grammars and uses this lattice to find regular grammars more efficiently than the straightforward enumerative methods. Constructive methods build a plausible grammar using only positive examples. Gold [Gold67] proved that it is impossible for a program to determine the grammar, in a finite amount of time, for a context free language when the program only receives positive examples. Although impossible to find the exact grammar, heuristics have been found to guide the process in finding a "good" grammar. Some of these heuristics are based upon the *distribution* of substrings in the language. Refinement methods refine

a hypothesis grammar as new examples are presented to the system. These refinements include merging and simplifying.

The PLAND system uses components from both the constructive method and the refinement method. The distribution heuristics of the constructive method are analogous to heuristics used by PLAND to find the largest loop and conditional structures. The simplification of the refinement method is the same as the subsumption check performed by PLAND when it finds a new macrop, see section 6.1. The methods listed here are for complete grammars, but the processing done by PLAND when building the macrops is a combination of the constructive and refinement methods. Both methods must be used since the system has no examples from which to work. PLAND uses the constructive-like methods when building up a new substructure and uses the refinement-like methods when eliminating subsumed macrops and incorporating previously defined macrops in new ones.

3.6.2. SNPR

The SNPR system [Wolff82] discovers regular grammars from unsegmented text inputs. Wolff has proposed this system as a model for language acquisition. Although the domain for PLAND is very different, when PLAND is working without background knowledge it performs nearly the same task as SNPR. Both systems discover loops, conditionals, and sequences. SNPR's representation makes the grammar much more explicit than PLAND's representation. SNPR uses a concept of cognitive economy called *compression capacity* (CC). CC is a measure of the effectiveness of a grammar for compressing data. The compression of data is the driving force for Wolff's system, much as *cognitive savings* is for PLAND. There are a couple of important differences between the systems. SNPR uses a hill-climbing search strategy when developing the regular grammar. PLAND performs a best-first search when no background knowledge is present. Due to the hill-climbing approach, SNPR only finds one grammar for the input example, where PLAND finds many different grammars depending upon which macrops are considered part of the

machine. The multiple grammars of PLAND are not all simple variants. They differ through changes controlling which overlapping macrops are allowed in the system. Another difference between the systems is that SNPR does not explicitly look for loops in the grammar. As in the NODDY system, loops are a result of adding a conditional to a rule. Wolff presents a very good point to critics that contend his system does not always find the exact grammar used to generate the input example:

> Since there is an infinite range of grammars which are consistent with any text, we may ask why one or some of them should be judged more appropriate than the rest.

The same sentiment applies to macrop discovery.

3.7. SUBDUE

There is another system being developed to discover substructures by Holder, named SUBDUE [Holder87]. SUBDUE works on the problem of discovering substructures in a more general domain, a domain where more than one type of relation may connect nodes and where a node may have more than one relation of the same type. One can think of this as discovering substructures in a graph. The research of PLAND has focused on discovering substructures using background knowledge to direct search in a domain with one type of relation and only one possible connection to a node. The research on SUBDUE has focused on more general substructure discovery techniques, handling the problems of substructure homomorphism and isomorphism in the more general case, without incorporating as much domain knowledge.

3.8. Related Work Summary

The PLAND system is related to many previously developed systems. Some of these relations are stronger than others but recognizing all of them is important. The key difference between PLAND and these other systems is the approach taken to the discovery task and the use of background knowledge. None of these other systems (except SUBDUE) was specifically directed at discovering substructures. None of these other systems used background knowledge with empirical

techniques, to the extent that PLAND does. So even though PLAND is related to many systems, in the true Gestalt sense PLAND is more than the sum of these parts.

CHAPTER 4

SUBSTRUCTURE DISCOVERY

This chapter explains formal aspects of substructure discovery. Definitions are given which are used throughout the remainder of the thesis. In addition, objectives of and problems with substructure discovery are presented.

4.1. Definition of Substructure

A substructure discovery system is given an event that consists of nodes and relations between those nodes. In PLAND this is a list of actions. In order to define substructures for an input event, the event must consist of two or more primitive or rudimentary components that articulate in some possibly abstract way. If the example has no distinguishable parts then there are no substructures which can be discovered. In practice, most everything can be broken down into smaller components but one usually stops the decomposition process at some useful grain-size. The smallest grain-size to be used in the discovery of substructures will be a given and such entities will be called nodes. This does not mean that the nodes must denote some rigid description of an entity. Actual nodes of a system are fixed, but the system discovering substructures may generalize over the actual nodes and a previously discovered substructure may be considered as a node. But all nodes of a given type are considered equal, whatever matching criteria are used to determine the type of the node.

Nodes must be connected to create the whole object. Relations are used to define the ways in which nodes may be connected. Relations are predicates on nodes and may have any arity. However, relations defined on relations are not considered. In all examples encountered where

relations were defined upon relations, the predicate could be redefined as a relation on nodes. Consider the relation of *angle-between* on two lines. If lines are themselves relations between endpoints then angle-between(a,b), where a and b are lines, is a relation on relations. But this may be recast as angle-between-nodes(a_1,a_2,b_1,b_2) where the relation is now defined on the endpoints of the line relations and maintains a first order expression.

A substructure can now be defined as a collection of relations and the nodes associated with those relations. The nodes and relations constitute a connected portion of the structure within a complete event. All the nodes used by relations must be connected in the graph theory sense where the nodes are vertices and the relations are edges. Thus, a substructure cannot consist of two (or more) disconnected parts. Not all the nodes of a relation need be included in the substructure. Some nodes for a relation may be ignored. This allows for the substructure to be open ended manifesting a simple dropping condition generalization.

Structures are typically built from smaller structures by adding nodes and relations. Because of this, it is important to define the possible starting points for substructure discovery. The smallest substructure that may be defined is the null set. All structures start from this point, but the system does not work with it directly. The two smallest non-null structures are a single node and a single relation. A single node defines a simple structure upon which more complex structures may be built by adding relations and nodes. A single relation by itself is not a good starting point because implicit in the concept of a relation is the notion of endpoints or nodes. Such nodes are needed to define explicit occurrences of the relation, so a relation without endpoints is rarely used as a simple structure.

4.2. Objectives for Substructure Discovery Algorithms

This section describes what a domain independent, heuristic based substructure discovery algorithm should attempt to accomplish. Not all of these objectives are important in every domain. These objectives do not take into consideration the use of domain specific background knowledge.

Rather, these objectives apply to the base level algorithm before background knowledge is incorporated into the solution. It is important to understand how design choices made at the lowest level affect the algorithm when it is extended.

(1) The algorithm must be capable of generating all possible substructures expressible in the language it is using. The algorithm will probably be guided by heuristics to prefer certain types of substructures over others, but it should not be biased so as to ignore or not have the ability to represent any substructure.

(2) The algorithm should be able to use previously discovered substructures as components of a structure currently being constructed. By using recently discovered substructures during the next cycle of discovery the system uses advantageously the powerful savings of substructures.

(3) Identical substructures should be identified and not processed further. Although it is obvious that homomorphisms should not be processed more than once, in practice it can be difficult to tell that one is working with identical substructures. Consider figure 4.1, where in the midst of processing the components for a square the system attempts to define the square by two different ways. First, it is defined as 2 L shaped components connected by the ends. Second, the square is represented as 4 L's where overlap of the sides is allowed. Clearly, the newly

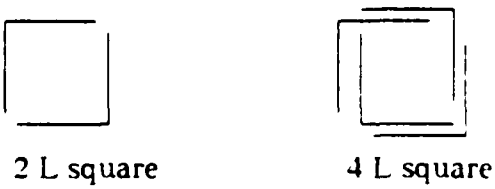


Figure 4.1: Homomorphisms for a Square

discovered substructures for the squares will occur in the same positions and will have the same possible extensions in future substructures.

(4) Isomorphic substructures should also be handled in an efficient manner. Again, as with homomorphisms, one wants to realize that the same substructure is being represented multiple times. However, with isomorphisms all variants of the substructure may not be equivalent when the substructure is grown. Figure 4.2 demonstrates this case. Assume the substructure discovered thus far is the square ABCD. Notice that it can be matched to the other square in a number of ways:

A->1 B->2 C->4 D->3

A->2 B->4 C->3 D->1

A->3 B->1 C->2 D->4

A->4 B->3 C->1 D->2

It is tempting to throw away all but one of these isomorphisms. This strategy fails when E is

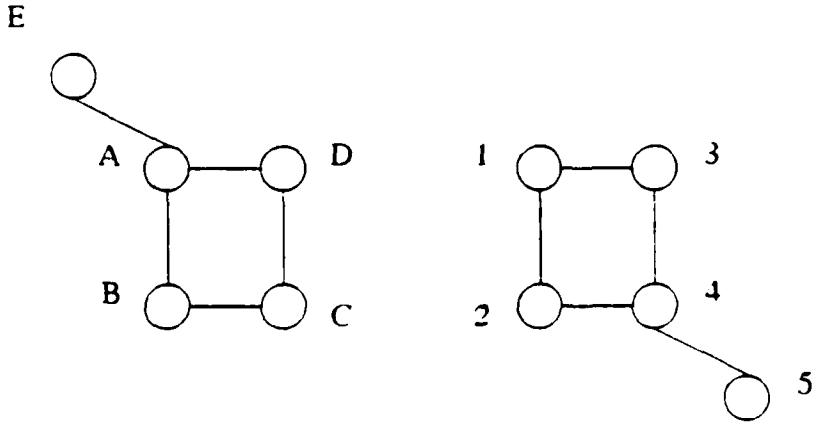


Figure 4.2: Isomorphism Example

added to the original square substructure. The only isomorphism which matches the grown substructure is the last one presented. It should be clear that this is a difficult problem.

(5) The algorithm should allow for overlap between substructures, but control it. Certain substructures may not be discoverable (or may be much more difficult to discover) when overlapping substructures are not allowed. Consider the string *ABCDEABCDEABCDE* as given, and suppose the system has already discovered *ABC* and *CDE* as useful substrings which it can reduce. If the system allows *C* to be used by both an occurrence of *ABC* and *CDE* then the input sequence can easily be reduced by the known sequences. If the concurrent use of *C* is not allowed the system must *compute* that *ABCDE* is a useful reduction of the string. While the use of overlap is extremely helpful, it is also difficult to control the complexity of the substructures discovered. Figure 4.1 gives an example of this.

An algorithm for discovering substructures does not need to have all of the components mentioned above but one should be aware of how the algorithm handles each of these situations. As with many issues in computer science there are trade offs between implementing each of the above goals and ignoring them.

4.3. Simple Algorithm

Presented in figure 4.3 is a simple, representation independent algorithm for discovering substructures. It does not address the issues of identifying isomorphisms and homomorphisms. The algorithm demonstrates the growing process of a substructure, which means taking already discovered substructures and extending them by one node or relation.

4.4. Representation Issues

As with most tasks in artificial intelligence there is not a clear cut representation to use for the discovery of substructures. A simple graph structure with nodes used as vertices and relations as edges will always work. But the algorithms to manipulate the graph structure can be

```
/* Given-example is the item for which substructures are being discovered */
Structure := nil                          /* structure currently working on */
Discovered-list := nil          /* list of found substructures */
To-process-Q := list of all smallest nonempty substructures (nodes)
                                  /* list of partial structures to extend */
Do while more items in To-process-Q
  Structure := Pop To-process-Q
  If Structure not homomorphic to an item in Discovered-list
    Find all occurrences of Structure in Given-example
    Push Structure with occurrences onto Discovered-list
    Extend Structure in all possible ways for each occurrence, using
      previously discovered structures when applicable
    Push each extension onto To-process-Q
  Endif
Enddo
```

Figure 4.3: Simple Substructure Discovery Algorithm

computationally expensive. For many domains a more efficient representation can be devised which will aid in the discovery process by making homomorphic and isomorphic structures explicit and thus easy to determine. There are other issues that can be used to help determine if a representation is adequate for substructure discovery:

(1) Extensible portions of the substructure should be easily identifiable. This facilitates the growing process.

(2) Substructures should be extensible by concatenating previously discovered substructures.

(3) Substructures should be usable as nodes. The importance of this depends upon the domain for the system but a greater range of substructures can be handled when embedding is allowed.

Another important facet of a representation is that both nodes and relations must be expressed in the representation, either explicitly or implicitly. A representation defining only nodes or only relations is not sufficient for substructures. Consider a representation expressing only nodes. Let the substructure being defined consist of two nodes with a single relation between them.

such as (N1 N2). If there are two relations between nodes N1 and N2 then this representation is ambiguous. Likewise, a relation needs its endpoints given to uniquely identify it. If just a relation type is given it matches any relation of that type in the given example.

4.5. Cognitive Savings

Cognitive savings is a value calculated to determine the usefulness of a discovered substructure. A system would use the cognitive savings values to determine which of two substructures had the best potential for extensions. The intent of this value is to capture the mental savings one gains by working with the substructure instead of the primitive actions that compose it. A simple formula for cognitive savings is

(number of structure occurrences - 1) * size of structure

where size of structure can be defined as number of nodes, number of relations, or some other formula using the components of the structure. This formula incorporates components of Wolff's compression principles [Wolff82]. A macrop that can chunk a large number of primitive actions and occurs many times is very useful. There is a trade off when growing a large macrop and some occurrences are no longer covered. The exact threshold for this cut off point is domain dependent.

Note that when extending substructures by other substructures and allowing overlapping components, one cannot merely add the cognitive savings values to achieve the value for the new structure. If one just adds the values then overlapping portions of the structures are weighted disproportionately. The cognitive savings for the new composite substructure must be recomputed.

A structure with optional parts or different sized components that are interchangeable can pose problems when computing the cognitive savings. There are at least three methods that could be used to figure the cognitive savings for conditional portions of a macrop.

(1) Minimum value. The choice point which results in the smallest cognitive savings could be used.

(2) Maximum value. The choice point which results in the largest cognitive savings could be used.

(3) Average value. A weighted average of the cognitive savings for all the choice points possible could be computed.

Determining whether the minimum value, maximum value, average value, or some other formula is used for the cognitive savings value is domain dependent.

CHAPTER 5

SYSTEM OVERVIEW

This chapter and the next one discuss the PLAND system. This chapter gives higher level information about the system such as the task accomplished, the goals of the system, and a general feeling for how the parts of the system work together. In the next chapter specific modules and how they function are presented.

5.1. PLAND's Task

The goal of the PLAND system is to discover macro operators (macrops) from a given trace of primitive actions. The system uses background knowledge to help guide the quest for macrops. Consider a robot that wants to learn how to pick up dirty clothes on the floor and put them in the clothes hamper, carry the clothes hamper to the washing machine, and do the laundry. This robot learns by observing others perform the task, then extrapolates from that observation what is useful. In the problem at hand, the person would pick up all the clothes in one room and proceed to the next room to pick up the clothes. By watching the actions of the person doing this, the robot could learn a macrop for picking up clothes (goto piece of clothing, grab clothing, move arm to basket, ungrasp clothing). It would then realize that this macrop is repeated many times for each room — a looping construct is discovered. The robot then notices that this loop is always embedded in the context of going to a room, performing the *pick up clothing loop*, and exiting the room. Another loop is formulated.

Discovering structure within given input events requires the use of *part-to-whole generalization*. In other words, from the pieces the system sees it explores how the complete event

is best described. There are two types of structure being found by the PLAND system. The linear structure of actions is composed into macrops. There is also the hierarchical structure of the macrops that can be used to break the task into manageable chunks. These chunks can be used to recognize the hierarchy of goals of the problem solver. The system works at discovering the first kind of structure, macrops, but the hierarchical structure is a byproduct of allowing macrops to internalize other macrops and is not discussed further.

Structure in this system is the relationship between actions used to accomplish the plan. Logical groupings of actions that perform a definable unit of work are assembled into macro operators. A single macro operator is a sequence of actions that are structurally linked by the order in which they are performed, a totally ordered subset of plan steps. The nodes of this structure are the actions and there is only one type of relation between actions: precedes or follows.

Unlike the macro operators of other systems [Fikes72, Minton85], macrops for this system are not generalized by changing constants to variables and determining all of the preconditions for the execution of the macrop. The PLAND system is concerned with discovering possible macrops that are known to accomplish some task in the given execution trace, but whose general applicability is as yet unknown. Macro operators discovered by the system could be passed to an explanation based learning (EBL) system [DeJong86, Mitchell86] to determine the macrop's usefulness and to be generalized. Usefulness could be determined by the "conciseness" of the proof tree for the macrop and the applicability of the macrop in achieving the higher level goal. Having PLAND propose discovered macrops to an EBL system means the system is deriving proofs only for macrops that have some statistical support. PLAND has taken a large sequence of actions and through various techniques discussed later, partitioned the actions into seemingly useful units. These units can now be passed as single examples to the EBL system. It would be intractable for an EBL system to attempt to discover the advantageous macrops from the initial action trace as there are too many possible combinations about which to reason.

PLAND does not deal with the preconditions of the actions it handles. The system is based upon similarity-difference based learning (SDBL) systems [Hayes-Roth78, Hoff83, Stepp84, Vere78]. However, this does not preclude learning of general macro operators. As with all similarity-difference based systems it does not try to prove that what it has discovered is actually a valid macrop. There is a leap of faith in the generalization. It is possible for the system to classify an anomaly as a macrop. But in order for this to happen under typical heuristic biases, the anomaly would need to occur many times. In such a case one must question how irregular the observed actions really are.

Consider the case where our learning robot has observed that everyone walking from point A to point B follows the path indicated by the arrows in figure 5.1. The knowledgeable robot is aware that the shortest distance between two points is a straight line. Yet by observing the pedestrians in the area it is clear that no one takes the shortest distance between points A and B. What would a prudent robot do in moving from A to B? If the robot were EBL based, the shortest path between the points would be taken: after all there is no information to indicate why any other choice is better[1]. A robot based on the PLAND system would follow the same path as the other pedestrians. Now if it happens that all the observed walkers were out for casual strolls and had no time constraints, this might be foolish. But if there was a "keep off the grass" sign unobserved by the robot, or worse, the rectangular region was a mine field unknown to the robot, then the path of the others was indeed appropriate.

The PLAND system has a single sequence of observed actions as input. From this stream of actions it must discover logical units that can reduce the complexity of the trace. This is similar to the NODDY system of Andreae [Andreae85], but there the problem is one of learning from examples. PLAND must break the action sequence into what it believes are examples and work with those chunks to discover macrops. This complicates the problem because one is never sure that one is working with the correct "examples."

[1] This occurs due to incomplete domain knowledge in the EBL system.

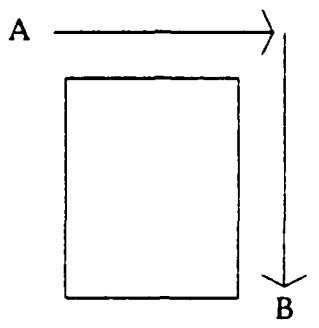


Figure 5.1: Observered Path from A to B

5.2. Types of Macrops Discovered

Three types of macrops are discovered by the PLAND system: sequences, loops, and conditionals.

- Sequences

  The most basic macrop is a simple sequence of steps. A sequence is a block of actions that have been used in many places in the input trace. This sequence has not occurred consecutively enough times to be considered a loop. A non-looping sequence is the most difficult type of macrop to discover. PLAND keeps sequences it has found during processing in a separate category from the other macrops discovered. These sequences are called partial macrops. The system requires background knowledge to promote a partial macrop to a "complete" macrop. If this where not the case then the system would quickly be searching through so many macrops that it will have trouble discovering new macro operators[2].

[2] The reason for this will become clear in the next chapter where the details of the routines are explained.

- Loops

  Loops are defined as sequences that appear juxtaposed for at least a minimum number of iterations. In the normal meaning of the word, loops have test conditions to stop their execution. PLAND does not determine what those stopping criteria are but learns only the sequence of actions that compose the body of the loop. If it so happens that the stopping condition for the loop is a perceptible action, then the system will discover a macrop that encompasses the loop macrop along with the stopping action. In general, learning the exit conditions for a looping construct require a system like BAGGER [Shavlik87a, Shavlik87b] since the conditions are reflected in the state of the system but not the actions performed. To clarify what is meant by a loop, consider an input string of *ABCDCDCDEFCDCD*, then the loop macrop, using formal grammar syntax, *(CD)** is discovered.

- Conditionals

  The third type of macrop found is conditionals. A conditional allows a choice of actions for some particular point in time. PLAND defines conditionals as macrops that have more than one choice point within them. A particular choice point is not limited to just two alternatives. As a simple example of conditional discovery, if given *ABCADCADCABCADC* the system discovers the macrop *(A(B+D)C)**. In a manner analogous to loops, the situations that cause a specific branch of a choice point to be performed are not learned.

These are the three types of macro operators that the system can discover. By nesting previously discovered macrops within other macro operators the system is able to discover complex relationships between different macrops and build up a hierarchical structure of an observed sequence of actions. The code to discover these constructs is the major portion of knowledge built into the system. Other heuristics and generalizations done by the system are supplied by background knowledge.

5.3. Top Level Organization

This section describes the top level data structures and control mechanisms. This information is required to understand the section on background knowledge which follows. The components of the system are described in this section and how the components work together is given in the section on background knowledge. The operations conducted on these data structures are deferred until section 5.4 because the domain knowledge plays a key role in the operation of the system.

The system works on multiple levels of generalization called *contexts*. A context contains all the information needed to process a set of actions for a given level of abstraction. This includes the input sequence of actions, the previously discovered macrops, the agendas (explained later), the partial macrops (the sequence blocks not yet believed to be macrops), and information about how macrops overlap and subsume each other. The system can proceed to a more abstract level by creating a new context in which the actions are generalized. The actions can be generalized by replacing groups of actions with macrops or by the use of a fuzzy matching algorithm. This matching algorithm allows flexibility in determining which actions are considered equal. The equality test among actions is important since that is how the system determines if a sequence is repeating. The fuzzy matcher can work at multiple levels of match such as forcing actions to be identically equal ("eq equal"), ignoring certain parts of the action, or considering all actions equal. When it is determined that a new level of generalization is required, the matcher is run on the actions to produce more general actions which form the basis of a new context. Macrops that replace actions in the original sequence are treated as actions in the new context.

A context is a self-contained data structure. The system can change the level it is working on by replacing a variable with a different context. The flexibility of this scheme is useful if the system is unsure which level of generalization is appropriate for the problem. The system can work on one context for a specified amount of time then swap contexts and work on a different one.

The other top level data structure is an *agenda*. The agenda indicates where to look for new macrops in the given example. There are many agendas competing for processor time, and a simple agenda control system manages their priorities. An agenda contains information on where in the action sequence the search for a macrop is to begin. The previously found macrops that can be used in building up the current macrop are specified in the agenda. The type of macrop wanted is indicated, either a loop or a conditional. Searching directly for sequence (partial) macrops is not done, but information regarding a possible sequence macrop is updated whenever a new macrop is discovered. The agenda also has other information specific to the type of macrop specified. Agendas compete only with other agendas in the same context.

Agendas are created in a variety of circumstances. The initial agendas indicate that the system should search for a loop or a conditional form starting with the first action of the input sequence. These are the default agendas produced by the system. As agendas are processed they spawn new agendas. If an agenda fails, the start position is incremented and in the case of loops the "skipping factor" is incremented. The skipping factor tells the loop finding module how many occurrences of the action at the start position should be bypassed when looking for the body of the loop. If the agenda is successful a macrop has been discovered, and two new agendas are created which use the discovered macrop. The new agendas use the macrop just found in addition to the other macrops available during the discovery, as specified by the agenda. The new agendas start searching at the beginning of the input sequence. One of the new agendas is for loop finding and the other is for conditional discovery. When the new agendas are added to the agenda list, other agendas which they subsume are removed. Subsumption means that the old agenda can only use macrops that are also usable by the new agenda. Not all agendas are subsumed by the new agendas. If two macrops happen to be in conflict, that is both describe a common instance of actions, then agendas containing either macrop must be retained. The number of agendas quickly explodes and additional knowledge is needed to control it.

5.4. Use of Background Knowledge

The AI community has recognized that in order to learn substantive concepts a system must possess knowledge about the domain [Schank86, Winston84]. Previous SDBL systems [Fisher85, Hoff83, Langley86, Stepp84] have not used background knowledge in a flexible manner to guide the searching process, but rather have used knowledge to control the generalizations allowed. Previous SDBL systems used a fixed set of knowledge to help guide search whereas PLAND can use different aspects of the knowledge depending upon the current context. Recently, researchers have incorporated more background knowledge into the systems to help with the discovery process [Lebowitz86, Mogensen87, Stepp86]. PLAND continues the trend of the latter. This section describes the multiple ways in which domain specific knowledge is used by the system.

In the current version of PLAND the background knowledge is expressed by rules. The system does backward chaining through the rules to obtain an answer to a query. The current method of retrieving the knowledge is not very efficient. However, this is not the focus of the research. What is important is how the system uses the knowledge given, not the representation or access method of that knowledge.

5.4.1. High Level Background Knowledge

PLAND uses background knowledge in three distinct ways. At the highest level, the background knowledge performs the function of meta-knowledge. The system works on levels of generalization called contexts (previous section). A context contains all the information needed to find macrops at a given level of abstraction. This includes macrops that are already used and the generalized action sequence. After an agenda has been executed, the background knowledge is inspected to see if the current context is still preferred over another. If a different context is wanted, the knowledge base simply returns the new context to be used. At this level the knowledge is used to control the level of generalization of the action steps. Thus the system can process the input at a higher level of abstraction after some macrops have been found. If the search

at the higher level is fruitless then the system can return to the lower, more detailed level to attempt to discover more useful macrops.

This is a powerful mechanism because it allows the system to pursue many possible goals. If the system discovers a macrop that indicates a certain environment is present, then it can create a context that generalizes some of these actions to help confirm that notion. For example, if the system discovers at the base level a "picking up cans" macrop, it might try to confirm that the current environment is a grocery store. This could be accomplished by searching for a macrop that pushes a shopping cart, and generalizing cans, bottles, boxes, and bags to be considered equal in a new context. After searching in this generalized context for grocery store macrops without success, the system can return to the original context to look for more macrops. Working at the primitive level it then might discover a macrop for "pulling weeds", so using the knowledge of picking up cans and pulling weeds in combination one suspects the task is cleaning up a yard or a roadside. There may be more than two competing contexts at a time. The system could also pursue both the grocery store and the cleaning up the yard ideas at the same time.

When the top level decides to change contexts, background knowledge is consulted as to what type of generalizations should be performed on the actions of the current context. Consultation with the background knowledge in this fashion allows the system to discover macrops that would be impossible to discover otherwise. A system without knowledge specific to the domain could not make logical guesses at which of the many possible generalizations has meaning to the problem at hand.

5.4.2. Medium Level Background Knowledge

The background knowledge used at a lower level helps direct the searching process for the macrops through agenda control. Before any agenda is given control, the background knowledge is consulted to approve its applicability. The agenda usually is approved and executed. If the knowledge indicates that macrops are not to be searched beyond a certain point in the input

sequence. then those agendas can be pruned. Information contained in the agenda could signify that the agenda should not be performed. The knowledge used in this method can save substantial amounts of processing and significantly prune the search tree.

5.4.3. Low Level Background Knowledge

At the lowest level. background knowledge is used to control the macrops allowed by the system. After finding the sequence of actions for a macro operator. background knowledge is consulted to determine if the sequence meets any simple criteria expressed for macrops. With simple rules for checking macrop sequences the system is able to eliminate the generation of unuseful macrops. This saves not only storage. but also the expense of handling agendas with the new unproductive macrops and the searches they generate. The example in section 7.2.2 is a case where this type of check allows the system to discover useful macrops. when it could not without the information. After all the occurrences of a macrop have been identified. a second reference to background knowledge is made. This time the information is used to confirm the validity of the macrop. Now with the information of exactly where the macrop occurs. the usefulness of the macrop can be better evaluated and if not as great as first suspected. the macrop is eliminated.

Determining which partial macrops (the sequence blocks found up to date in the context) should be promoted to useful macrops depends upon given knowledge. There are many partial macrops generated; any sequence of actions that occurs more than twice in the input is a potential useful sequence. All such sequences must be kept because as macrops are discovered. the partials can become more important. However. to convert all those sequences to macrops would bring the system to a halt in its search for other useful macrops. Knowledge can be used to promote only those partial macrops with some great number of occurrences or that accomplish some specified task.

Knowledge at the lowest level can rid the system of macrops having low utility. Although this seems trivial. this level of control determines whether a system finds a solution or runs out of

space and/or time. Control like this is missing in many SDBL systems. Those systems can only make guesses at what is useful, much as this system does when no background knowledge is present. The implication that nothing valid can be done without background knowledge is not intended. In fact this system can find some useful results even when no knowledge is given. In these cases the system acts like a finite state machine builder. The input is like a string where the actions are the letters of the string. Then the discovered macrops act as formal grammars defining portions of the input string. Taken together they constitute a generalized regular grammar that can generate the input and other "similar" strings.

5.5. Representation of Macrops in PLAND

Macrops are the structures discovered in PLAND. In the terms of chapter 4, the nodes are the actions and the one type of relation handled is "follows". The structure for a macrop is expressed as a list. In this representation a sequence is a list of the actions in the block that constitute the macrop. For a loop, a list of the actions in the body is the macrop. For a conditional, a sublist of actions represents a choice of actions at that point (one of the set may be picked.) In these representations the "follows" relation is implicit. This works because there is only one place where a node can be connected to another node. An action can only follow another action sequentially. If there were more ways in which actions could be connected or more than one type of relation then the relations of the structure would need explicit representation.

The data structure that defines macrops contains more than just the list of actions defining the macrop. It also contains the positions of the macrop in the given action sequence. Since macrops are linear structures that are uniquely defined by the position and the sequence of actions. These are the two major components of the representation. The system also maintains individual start positions for each occurrence of the loop body, and the length of each occurrence. Just because the same sequence defines two macrops does not mean they are of the same length. Different lengths are produced by conditionals where different choices have different lengths.

CHAPTER 6

SYSTEM DETAILS

The goal of this research in the beginning was to determine if discovering substructures in plans was even possible. The search for methods of discovering substructures has been transformed into the search for more efficient methods of discovering substructures. As expressed earlier, this system follows in the tradition of other similarity-difference based learning systems. Unlike previous SDBL systems, the soul of PLAND is not concerned with consistency and completeness statistics or the ratio of inter-cluster similarity to intra-cluster similarity, though these concepts are the strong roots that support the ideas given. Rather, this system strives to discover chunks of input sequences that when considered as a nonseparable unit increase the efficiency of understanding the input. Cognitive savings is the measure which represents the usefulness of a discovered macrop.

This chapter consists of four sections. The first three describe the major macro discovery procedures used by the system for loops, conditionals, and sequences. The sections are presented in this order for two reasons. Historically, this is the order in which the modules were created, and they logically build upon each other in this fashion. The fourth section of the chapter presents the details of the fuzzy matcher algorithm used for the generalization of the action steps.

6.1. Loop Discovery

This section describes in detail how loops are discovered in the PLAND system. Parts of this section describe procedures that are used in discovering other constructs in addition to loops but they are described only here.

6.1.1. Approach

The most basic concept underlying loop macrop discovery is *if a sequence occurs many times, with one occurrence following the other, then reduce the sequence.* This is a simple concept which has been used before to reduce given sequences [Restle70, Simon63]. But even this simple concept is difficult to implement in practice. Finding answers to simple questions can explode in exponential time when the examples are not explicitly given. Such questions as "where does the loop body begin," "how long is the sequence of the body of the loop," and "does the action (letter) that begins the loop also occur within the loop body (thus not always indicating a new iteration)" are difficult questions to answer. Yet they must be answered in order to discover the loop in the example.

The loop discovery module in PLAND expects parameters in the agenda to guide the search for answers to these questions. An agenda has three pieces of information used by the looping routines: where to begin the search, the number of iterations of the (proposed) body that must occur, and the skipping factor. By being parameter driven, the loop discovery routines can be focused to look at very specific areas, while retaining needed flexibility.

The search for the loop body begins at the position indicated. If no loop starting at this position is discovered then a new agenda is created that contains an incremented start position. A second agenda is created with an incremented skipping factor. The routines only look for a loop starting at the position given. Domain knowledge could be used to focus the search in specific areas of the input example.

The number of iterations of the loop body required to define a loop allows the system to control the level of generalization performed. The system always makes a leap of faith when it converts multiple juxtaposed occurrences into a loop. It is possible that there really is no loop at that point. This parameter allows one to make large leaps of faith by requiring only two iterations to appear consecutively. Likewise, requiring more iterations to occur before assuming a loop indicates a more conservative approach. Again background knowledge can be used to control this

aspect of the system, allowing information of the specific domain under consideration to determine what constitutes a valid loop. The use of this parameter allows a user to start searching for loops that have many occurrences of the body, thus having strong confidence in the "loopness" of the procedure found, and by decreasing the parameter value having the ability to find loops in which less confidence remains.

The skipping factor allows the system to discover loops where the first action of the loop body occurs later in the body. The skipping factor tells the system the number of occurrences of the first action to ignore in the body of the loop. This is important since the system looks for the loop body between instances of an action. For instance, given the input *ACABACABACABACAB* without being allowed to skip an occurrence of *A* the system could not discover the loop body *ACAB*.[1]

6.1.2. Schematic View of Algorithm

This section describes in a schematic way the algorithm used for loop discovery. Figure 6.1 presents the loop algorithm. A list is created that has the start positions within the example required to determine the loop body. As long as the sequence generated thus far does not completely describe the actions between two starting positions of the proposed loop then the sequence is extended (grown). The sequence is grown by all possible macrops and single actions. This does not explode, as there is a fixed number of macrops which are usable (as defined by the agenda), and each iteration of the proposed loop body acts as a constraint on what is allowed.

When a complete sequence has been found, background knowledge is consulted for approval. If it is accepted, the discovered body is compared to other discovered macrops to insure it is not a homomorphism. This is a simple check on the sequence value of the loop body to insure that duplicate macrops are not introduced into the system. This test is fast and simple due to the constraints and representation of sequences in the plan sequence domain. For discovering

[1] The string could also be defined by a loop with a body of *A(C-B)*.

```
1) Push list of start positions for body of loop on Q
2) Do while more sequences on Q
3)   If sequence is complete
4)     If not an old macrop and BK indicates macrop okay
5)       Build the macrop occurrence
6)       If macrop subsumed by previous macrop
7)         Store macrop on unused list
8)       Else
9)         Put macrop on active list
10)        Make agenda to use new macrop
11)      Endif
12)    Endif
13)  Else
14)    Extend sequence in all possible ways and push
         these sequences on Q
15)  Endif
16) Enddo
```

Figure 6.1: Algorithm for Discovering Loops

substructures in general this is not necessarily true. When it is determined that the new macrop is not a homomorphism of another, other occurrences of the macrop are found in the input example.

After the structure for the representation of the macrop has been built the subsumption test is made. For this, the body of the loop macrop is converted to a finite state machine representation. By using standard routines for building the complement of a machine, performing the union of two machines, and checking intersection of finite state machines [Hopcroft79], this macrop is checked against the other macrops of the system. The goal here is to eliminate macrops that are subsumed (can be generated by) other macrops. The new macrop may subsume some previous macrops and may be subsumed by others. A data structure like an ATMS [de Kleer86] is used to maintain the relationships between macrops and to indicate which macrops subsume which others. Note that subsumption testing can be very expensive, not only in this domain but in most domains in which substructure discovery happens. In general some type of graph matching algorithm is required.

6.1.3. Overlap Detection

When macrops are built from the discovered sequence, there is a check for overlap. Overlap of macrops is different than subsumption. When one macrop is subsumed by another it is completely covered or defined by that macrop. When overlap occurs only a portion of each of the macrops is covered by the other. PLAND does not allow overlapping macrops to be used in the same agenda. The ATMS-like structure used to record subsumption dependencies is also used to track the overlapping macrops. An overlap between macrops is indicated by a contradiction between the two macrops.

Currently, a crude method is used to determine overlap. The start positions and length for each occurrence of the macrop are compared with those values for other macro operators in the system. This is a very expensive test. A graph matching algorithm which recognizes overlap could also be used in some domains. For the plan sequence domain, however, this is difficult. PLAND finds overlaps which actually occur, and not overlaps that might potentially occur. For example the macrops *ABC* and *CDE* could overlap on *C* and a graph matching solution would find such an overlap. However, in the example presented to the system this might not ever happen.

6.2. Conditional Discovery

The discovery of conditionals is more complex than the discovery of loops. The problem with discovering conditionals is that anything could be made optional. In the extreme case a sequence could be described by a loop of length one with the body consisting of a single conditional for all possible actions. The algorithm used by PLAND avoids this pitfall by requiring that all conditionals have a base or key point that cannot be part of a choice set. An overview of the conditional discovery algorithm is given in figure 6.2. The basic principle in discovering conditionals is to find actions that occur on fixed intervals, then make conditionals out of what lies between these key points.

1) Compute difference arrays for items in a sequence
2) Compute the number of juxtaposed differences of equal value
3) Find the item with the largest number of occurrences
4) Fill in the sequences around the key by delta - 1 actions on each end
5) Return sequence which minimizes number of elements in the conditionals

Figure 6.2: Algorithm for Discovering Conditionals

The first step in the discovery of conditionals is to build a difference array. The difference computed is the number of actions between two sequential occurrences of an action type. The system uses all of the macrops that are passed in the agenda. Because of the way the differences are computed, overlapping macrops cannot be allowed. Consider the problems of allowing overlap among two macrops. There is no a priori method of determining which of the overlapping macrops should be applied. And by applying one of the macrops in the wrong position, the pattern for the conditional could be lost. The problem arises only if the overlap actually exists in the observed sequence.

Null conditions are found indirectly by the system due to the way differences are computed. If a null branch was allowed then there could be any number of actions (either real or null) greater than the real number of actions between any two action type occurrences. This implies that there could always be a conditional structure discovered — just introduce enough choice sets with null so that the largest difference between two sequential action types is covered. The other differences will be described by using the null choices. Clearly this is not what a conditional discovery module should do. PLAND can handle choice sets with null, however background knowledge must indicate that a particular action is optional.

Step two is to build an array which indicates the number of consecutive equal differences for each difference array. This is done so that in step three the largest iteration can be found. From the explanation thus far, it should be clear that the conditionals discovered must be part of a loop. In

fact, the conditional found is a loop body. It is the repetition of actions at a fixed distance from each other in the sequence that allows the conditionals to be discovered. The actions do not have to be a fixed distance from each other in the primitive version of the observed trace. But they must be a fixed "action" distance apart which means a variable length macrop (such as a loop) could be used in the conditional.

This algorithm cannot discover the choice set (AB + C) directly since the length of the elements in the choice set varies. Again this is due to the way the algorithm depends upon distances between the key points. This choice set can still be discovered, however, by using the partial macrop discovery process described in the next section. If AB is made into a macrop, say A-B, it could then be used to discover the choice set (A-B + C). Through the application of partial macro operators, all conditionals that would be discoverable by allowing variable length items in the choice set are still discoverable.

Now that the key points of the conditional have been found, the fourth step of the algorithm fills in the steps around the key. This is where the actual choice sets get constructed. The delta described in the algorithm refers to the distance between occurrences of the key. There are actually many positions within the conditional macrop where the key element could be placed. The key item could be the first action of the macrop, the last, or any other in between. This algorithm uses a simple heuristic of trying to minimize the number of values in the choice sets. The choice sets are built up from all the items that are the same distance from a key point.

The last step of the algorithm is to convert the best descriptions of the conditional into macrop structures and find the other occurrences of the conditional. If there are ties for the best conditional then all of them are returned. The system returns agendas which use the newly discovered conditional macrops. Figure 6.3 demonstrates a simple example of conditional discovery.

Input sequence:
a b c a b c a d c a b c a d c

Difference arrays:
(a 3 3 3 3 start 0)
(b 3 6 start 1)
(c 3 3 3 3 start 2)
(d 6 start 7)

Juxtaposed differences:
(a (4 3) start 0)
(c (4 3) start 2)

Select best differences:
(a (4 3) start 0)

Build possible conditionals:
(()+b+d) (()+c) a (b+d) c

Select sequence that minimizes conditional length
a (b+d) c

Figure 6.3: Example Conditional Discovery

6.3. Sequence Discovery

Although the algorithms for the discovery of loops and conditionals are powerful, alone they are not enough for a plan sequence discovering system. When there are sequences of actions that occur many times in an observed plan, savings in complexity can be realized by replacing a repeating block by a single item. This can happen even when the block sequences do not appear adjacent to each other [Schuegraf74]. Although blocks of actions are important to discover, there do not seem to be simple heuristics to follow in determining what constitutes a good, isolated sequence. This section describes how the PLAND system discovers these noncontiguous sequences of actions. As explained earlier in the thesis, these groups are called partial macrops as they are being built. The name partial macrops is appropriate because at any time they could be converted

to a macrop. Background knowledge is used to determine which partials should be converted. The name is also fitting as these macrops are never truly finished. As new macrops are discovered old partial macrops can be extended to become more useful partials with an increased cognitive savings value.

Partial macrops are created when the system is initially given an input observation. All combinations of macrops that occur sequentially are composed. If there is only one occurrence of a sequence, it is dropped as it can never become useful. As new macrops are discovered, they are used to extend all the blocks that end in positions where the macrop begins. When the old partials are extended they are not dropped, since new macrops may also start where they end. Instead, new partials are created. As a partial is extended, its number of occurrences may decrease or remain the same but never increase. An extension may not be applicable to all instances of a partial so some do not get extended, but no instances are ever added because of an extension. Extensions can quickly be applied because partial macrops are indexed by the next positions of all their occurrences. After a new macrop is discovered, the start position for each instance of the macrop is matched against the ending position of the partials. A partial may be extended by the macrop when a match happens.

All the partials must be retained because a macrop discovered in the future could increase the usefulness of the partial. However, limits may be set on the number of instances required for a partial to ever become a macrop. If the number of occurrences is below this threshold, the partial will not be added to the system. The processing of partial macrops is expensive in terms of space, because so many must be maintained to discover the few that are important. The current length of the partial cannot be used to trim unneeded partials from the system since unlimited growth (except by the size of example) may extend the partial to the required length.

Another pressing question concerns when a partial macrop should be converted into a macrop. There are two conflicting issues here. First, as more macrops are defined in the system more time is required to discover other macrops. In order to discover loops, for example, the growing process

must extend its sequences by all macrops that apply in a given position. If there is a useless macrop described for that position, extending the macrop with it causes wasted effort. Thus, there is incentive not to convert any more partial macrops than needed. On the other hand, there may be missed opportunities if a useful partial is not converted. Certain loops and conditionals cannot be discovered unless pertinent macrops are defined. For these reasons, domain specific knowledge is used to determine which partials should be converted into macrops. The background knowledge could use any criteria to select convertible macrops, including length, use of a particular action, or proving a macrop accomplishes some desired task.

6.4. Fuzzy Matcher

The fuzzy matcher is used to generalize actions for new contexts. The fuzzy matcher is a pattern based algorithm that operates in two modes. In one mode, the algorithm compares an action to the pattern and indicates when the action is an instance of that pattern. The other mode allows the system to create a more specific pattern. A general pattern and an action are again given, but a pattern is returned which has certain values changed due to values in the action.

A pattern is a set of directives used to determine what is a valid match. The commands allowed in a pattern are given in figure 6.4. Three commands are used exclusively when the matcher is creating a more specific pattern. They are :initial, :delta, and :deltar. These options cause the newly constructed more specific pattern to have bindings to corresponding parts of instances of the original pattern. This generates a pattern that must match at these locations exactly or to within some interval, by introducing :exact and :range directives respectively. The :initial directive generates an *:exact v* directive in the constructed pattern, where *v* is the value of the action for that slot. The :delta and :deltar commands generate *:range* commands that are based on the values of the action in that location of the pattern.

An example will help clarify how these patterns operate. Let an action of the observed trace be of the form (move x-position y-position) which indicates where the robot should be positioned

The meanings for the possible types of fuzzy match are:

:dc1	- do not care for one position
:dc+	- do not care for any number of positions - there must be one
:dc*	- do not care for any number of positions - null okay
:initial	- no value required to match pattern, but when an instance is create that value of the original is made into an exact match
:exact v	- there must be an exact (equal) match to the value v
:range (l h)	- in order to match the numeric value must be between l and h
:delta n	- an item will match an instance if it is = to the initial value or it is = to the value + n
:deltar (l h)	- like range but the range is based on the initial value

Figure 6.4: Commands for Pattern Matcher

at the end of executing the command. If the system wants all the move commands with the same x-position value to be considered equal, the general pattern would be (:exact move :initial :dc1). When this is passed to the fuzzy matcher algorithm with the action (move 15 35) the pattern (:exact move :exact 15 :dc1) is returned. This pattern would equate all moves to the x coordinate of 15. Similarly a range could be established.

When the PLAND system is building a new context, patterns introduced through background knowledge are used to generalize actions. If no patterns are introduced the actions are carried to the new context unchanged. If patterns are specified then an action is first tested to see if it matches any working pattern already created, thus allowing it to be considered equal to all other actions that also match that pattern. When it does not match a working pattern the action is passed to the matcher with the general patterns in an attempt to build a new working pattern. If this also fails the action is unchanged in the new context.

CHAPTER 7

EXPERIMENTS WITH PLAND

The capabilities of PLAND will be illustrated by several examples of its use. The examples run to illustrate the PLAND system may be classified into two major categories. First, there are examples that do not use background knowledge: when these are run the system discovers a regular grammar that generates the input string. PLAND in this mode operates mostly like previous similarity-difference based systems. The second category of examples uses background knowledge to help guide the search for macro operators. The domain knowledge is able to differentiate between useful and unproductive macrops that have been discovered. In this chapter examples of each type are given.

7.1. Examples without Background Knowledge

This section explains three examples which do not require any domain knowledge. The first example is presented with a walk through in multiple steps to facilitate the reader's understanding of the system. In order to emphasize the point that background knowledge is not used, the actions are given as letters. The input to the system is a list of actions and each action is a list, since normally there is more than one component to an action.

7.1.1. Example 1

Figure 7.1 shows Example 1 as it would be entered into the system. When looking for ways to reduce the complexity of Example 1, one notices a number of contiguous X's that implies X^* could be a macrop. The system discovers the pattern and replaces (conceptually) the items by the

((A)(B)(Y)(X)(X)(X)(Y)(X)(X)(Z)(Y)(X)(X)(Y)(X)(X)(X)(X)(Z))

Figure 7.1: Example 1

discovered macrop, changing Example 1 to get the sequence of figure 7.2. At this point it becomes apparent that the sequence can be reduced by a proposed macrop (Y X*)* as shown in figure 7.3. The macrop just discovered followed by Z also may be reduced by forming a macrop. This is done by PLAND but not illustrated in a figure.

The previous walk through demonstrates how PLAND systematically reduces the input string when discovering macrops. The system does not actually replace the contents of the input but marks where all occurrences of the discovered macrops are. When searching for a new macrop, old macrops are used whenever possible. The actual output of the system for this example is given in figure 7.4. As macrops are discovered the system assigns a name to them. Macrop names take the form M:c.n where c is the name of the context where the macrop was discovered and n is a two digit number indicating the order in which the macrops are discovered. In addition to displaying

((A)(B)(Y)(X*)(Y)(X*)(Z)(Y)(X*)(Y)(X*)(Z))

Figure 7.2: Example 1 with X*

((A)(B)((Y X*)*)(Z)((Y X*)*)(Z))

Figure 7.3: Example 1 with (Y X*)*

```
Observered trace of actions working with:
(A) (B) (Y) (X) (X) (X) (Y) (X) (X) (Z) (Y) (X) (X) (Y) (X) (X) (X) (X) (Z)
Ready to start another cycle.  The result of the last context was:

        CONTEXT 1
cogsav macrops
8.5    <M:1.03: ([M:1.02]* Z)*>
11.25  <M:1.02: (Y [M:1.01]*)*>
10.0   <M:1.01: (X)*>

The most interesting Partial macrops
12.0   <P: X X>
11.25  <P: [M:1.02]*>
11.25  <P: Y [M:1.01]*>
10.0   <P: [M:1.01]*>
Observered trace of actions working with:
(A) (B) [M:1.02] (Z) [M:1.02] (Z)
Ready to start another cycle.  The result of the last context was:

        CONTEXT 2
cogsav macrops
2.0    <M:2.01: ([M:1.02] Z)*>

The most interesting Partial macrops
2.0    <P: [M:1.02] Z>
Observered trace of actions working with:
(A) (B) [M:2.01]
All interesting macrops were discovered.  This example is finished.
```

Figure 7.4: Output of PLAND for Example 1

the macrop name, the sequence associated with the discovered macrop and the cognitive savings (cogsav) value are shown. The computation for cognitive savings is

(number of macrop occurrences - 1) * length of macrop

as discussed in section 4.5. The most interesting partial macrops found in this context are also displayed. Since this example runs without background knowledge none of these partials will be promoted to "complete" macrops (background knowledge determines which partials are promoted.) Notice that as macrops are discovered a partial macrop is added to the list. The new macrop is used to extend existing partials and as the starting point for new partials.

Before the system starts processing the next context the original sequence is replaced with the most interesting macrops. This example demonstrates that the most interesting macrop is used for replacement before others. If the longest macrop had been used instead of the most interesting (largest cognitive savings value) no macrops would have been discovered in the run of the second context as instances of M:1.03 are longer than those of M:1.02. The macrops of the first context are not passed to the second. They are placed in the sequence where appropriate and treated as action steps. Thus the system discovers M:2.01 which is the same as M:1.03 but it has no method of determining this. Example 1 demonstrates that PLAND is capable of using macrops within other macrops to discover nested loops.

7.1.2. Example 2

Figure 7.5 contains the output for running Example 2. This example demonstrates the system's ability to discover conditionals. The system discovers three interesting macrops. The first macrop found is the most interesting as it describes all of the string except for some *X*'s. The *X*'s were added as noise. Allowing *A* in the first choice set, in addition to being one of the key points, did not confuse the system. Along with finding the main macrop (M:1.01) the system also discovered two loop constructs. The system has noted internally that the loop of *A*'s overlaps with the conditional and thus the two macrops may not be used together. Only one context is generated for this example because all of the actions are covered at the end of processing the context. Without domain knowledge, the system uses this as an indication to stop processing. The macrop that gives the largest cognitive savings is a loop with the body *(A (B + C) D (C + E))*.

7.1.3. Example 3

The third example, whose run is shown in figure 7.6, demonstrates that conditionals may be found with embedded macrops. In this case the embedded macrop is a loop but its type is immaterial. The first macrop found by the system is a conditional but it does not yet have the

```
Observered trace of actions working with:
(A) (B) (D) (C) (A) (A) (D) (E) (A) (B) (D) (E) (A)
(A) (D) (C) (X) (X) (A) (A) (D) (C) (A) (B) (D) (E)
Ready to start another cycle. The result of the last context was:

        CONTEXT 1
cogsav macrops
1.0   <M:1.03: (X)*>
8.0   <M:1.02: (A)*>
20.0  <M:1.01: (A (B + A) D (C + E))*>

The most interesting Partial macrops
20.0  <P: [M:1.01]*>
8.0   <P: [M:1.02]*>
6.0   <P: A B D>
6.0   <P: A A D>
Observered trace of actions working with:
[M:1.01] [M:1.03] [M:1.01]
All interesting macrops were discovered. This example is finished.
```

Figure 7.5: Example 2 Output

embedded loop macro. This is because the loop macrop for X^* has not been discovered at this point; it is discovered next. Now the system finds the better conditional, M:1.03, which expresses the complete string, $((X^* + B)\ A)^*$. The system continues to find other macrops which are less interesting as indicated by the cognitive savings value. Notice that the macrop numbers are not sequential. Nonsequential macrop numbers indicate the system has discovered macrops that are subsumed by previously defined macrops. The subsumed macrops are noted and not used further by the system.

This concludes the discussion of examples that do not require background knowledge. In these cases the system finds regular expressions that define chunks of the input string. For all the examples shown the system made the largest jump of faith possible by requiring only two iterations of a string to constitute defining a loop construct. The system can discover conditionals and loops nested to any arbitrary depth.

```
Observered trace of actions working with:
(X) (X) (X) (X) (A) (X) (X) (X) (A) (B) (A) (X) (X) (A) (B) (A) (B) (A)
Ready to start another cycle.  The result of the last context was:

        CONTEXT 1
cogsav macrops
10.0   <M:1.10: (X [M:1.01]*)*>
5.5    <M:1.07: (A [M:1.02]* A B)*>
5.0    <M:1.06: ([M:1.01]* X X)*>
15.0   <M:1.03: (([M:1.02]* + B) A)*>
8.0    <M:1.02: (X)*>
10.0   <M:1.01: ((X + B) A)*>

The most interesting Partial macrops
10.0   <P: [M:1.10]*>
10.0   <P: [M:1.01]*>
10.0   <P: X [M:1.01]*>
10.0   <P: X X>
Observered trace of actions working with:
[M:1.03]
All interesting macrops were discovered.  This example is finished.
```

Figure 7.6: Example 3 Output

7.2. Examples with Background Knowledge

Background knowledge separates the performance of PLAND from other similarity-difference based systems. The two examples in this section demonstrate different uses of domain knowledge by the system. In the robot example, knowledge is used to indicate when a new generalized context should be constructed. The knowledge used in the second example, a mock trace of a typical week in a graduate student's life, prunes the search tree so that the correct macrop may be found. Without the knowledge in this example the system exhausted its resources before generating an answer during an experiment, as described in section 7.2.2.

7.2.1. Robot Example

In this problem, the system is given a trace of actions performed by a robot moving boxes from one room to an adjacent room. Figure 7.7 contains a map of two rooms showing the robot (circle), boxes, and the room layout. The robot moves the boxes from the room in which they reside to an identical position in the second room. The goal is to discover any useful macrops found from this trace. This requires the use of background knowledge. There are two important pieces of information given to the system. First, it is told that there is a doorway between positions (10,5) and (11,5). Second, knowledge indicates that if a doorway is used during the action performed, the robot is moving between two rooms. When more than one room is involved the system should create a new generalized context. In the new context the y coordinate of the move command is ignored and the object that is being grasped or ungrasped is ignored. These items are generalized to facilitate the search for macrops. The output from the program on this example is given in figure 7.8. The move command has the syntax (move to-x-position to-y-position), grasp has syntax

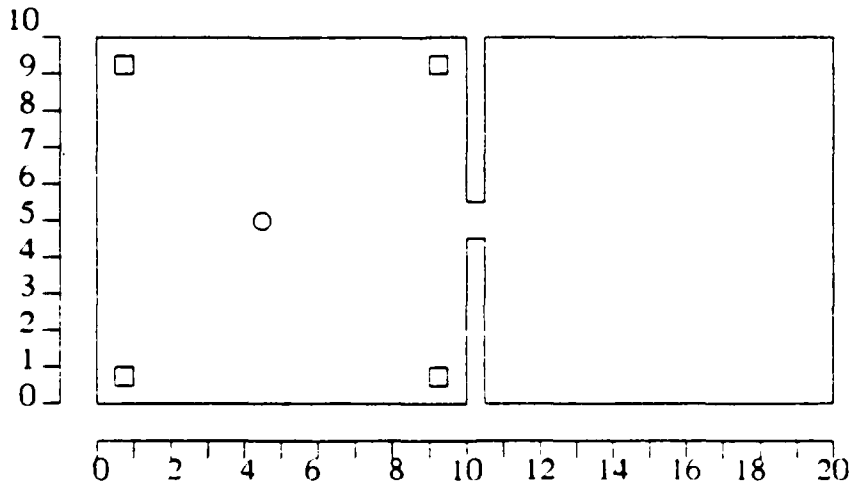


Figure 7.7: Robot Map

Observered trace of actions working with:
(START 5 5) (MOVE 0 10) (GRASP B1) (MOVE 10 5) (MOVE 11 5) (MOVE 11 10)
(UNGRASP B1) (MOVE 11 5) (MOVE 10 5) (MOVE 0 0) (GRASP B2) (MOVE 10 5) (MOVE 11 5)
(MOVE 11 0) (UNGRASP B2) (MOVE 11 5) (MOVE 10 5) (MOVE 10 10) (GRASP B3) (MOVE 10 5)
(MOVE 11 5) (MOVE 20 10) (UNGRASP B3) (MOVE 11 5) (MOVE 10 5) (MOVE 10 0) (GRASP B4)
(MOVE 10 5) (MOVE 11 5) (MOVE 20 0) (UNGRASP B4) (MOVE 11 5) (MOVE 10 5) (MOVE 5 5)
Ready to start another cycle. The result of the last context was:

CONTEXT 1
cogsav macrops
6.0 <M:1.02: MOVE-10-5 MOVE-11-5>
6.0 <M:1.01: MOVE-11-5 MOVE-10-5>

The most interesting Partial macrops
6.0 <P: [M:1.02]>
6.0 <P: [M:1.01]>
Observered trace of actions working with:
(START 5 5) {MOVE-0-DC*} {GRASP-DC*} [M:1.02] {MOVE-11-DC*} {UNGRASP-DC*} [M:1.01]
{MOVE-0-DC*} {GRASP-DC*} [M:1.02] {MOVE-11-DC*} {UNGRASP-DC*} [M:1.01]
{MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*} {UNGRASP-DC*} [M:1.01]
{MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*} {UNGRASP-DC*} [M:1.01] {MOVE-5-DC*}
Ready to start another cycle. The result of the last context was:

CONTEXT 2
cogsav macrops
6.0 <M:2.04: ([M:1.01] {MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*}
{UNGRASP-DC*})*>
6.0 <M:2.03: ({UNGRASP-DC*} [M:1.01] {MOVE-10-DC*} {GRASP-DC*} [M:1.02]
{MOVE-20-DC*})*>
18.0 <M:2.01: (({MOVE-0-DC*} + {MOVE-10-DC*}) {GRASP-DC*} [M:1.02]
({MOVE-11-DC*} + {MOVE-20-DC*}) {UNGRASP-DC*} [M:1.01])*>

The most interesting Partial macrops
8.0 <P: {UNGRASP-DC*} [M:1.01] {MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*}
{UNGRASP-DC*} [M:1.01]>
7.0 <P: [M:1.01] {MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*} {UNGRASP-DC*}
[M:1.01]>
7.0 <P: {UNGRASP-DC*} [M:1.01] {MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*}
{UNGRASP-DC*}>
6.0 <P: {MOVE-10-DC*} {GRASP-DC*} [M:1.02] {MOVE-20-DC*} {UNGRASP-DC*} [M:1.01]>
Observered trace of actions working with:
(START 5 5) [M:2.01] {MOVE-5-DC*}
All interesting macrops were discovered. This example is finished.

Figure 7.8: Robot Example Output

(grasp item), and ungrasp has syntax (ungrasp item). The braces in the output trace show that a generalized action has been created. The dashes separate the parts of the pattern used. For example {grasp-dc*} indicates that the first part of the pattern is an exact match to grasp and anything after that is ignored by the don't care indicator (dc*).

In the first cycle, the system discovers two partial macrops that the domain knowledge indicates should be made into macrops. These two macrops are for moving through the doorway (one from left to right, one from right to left). At this point the knowledge indicates that a new context should be built. The new context will use the new macrops and generalize the actions by specified patterns. The knowledge indicates the following patterns should be used to generalize the action steps:

(:exact MOVE :initial :dc*)
(:exact GRASP :dc*)
(:exact UNGRASP :dc*)

During the second context the first macrop discovered is for moving to a box (at either x position 0 or 10), grasping the box, moving to and through the doorway (M:1.02), moving to a position to set the box down (x position 11 or 20), ungrasping the box, and returning through the door (M:1.01). Two other less interesting macrops are also found. Without the knowledge to generalize the actions, the system would not have been able to discover the macrop. One point of interest is the system does not recognize the dependency of the x position in the move. Boxes that start at x position 0 in the first room are always placed at x position 11 in the second room. Also, the system does not understand that the last object grasped is the same as the object most recently ungrasped. An explanation based learning type system would be required to learn these dependencies.

7.2.2. Student Example

The input for this example, as shown in figure 7.9, is a week's worth of actions performed by a hypothetical graduate student. The goal of the system is to discover a macrop which will define a typical day in the life of this student. The background knowledge specifies that going to the gym is

```
Observered trace of actions working with:
(WAKE-UP) (EAT) (GOTO GYM) (GOTO WORK) (GOTO HOME) (EAT) (GOTO BED) (WAKE-UP)
(EAT) (GOTO WORK) (GOTO GYM) (GOTO HOME) (EAT) (GOTO BED) (WAKE-UP) (EAT)
(GOTO GYM) (GOTO WORK) (GOTO HOME) (EAT) (GOTO BED) (WAKE-UP) (GET-SNACK)
(GOTO BED) (WAKE-UP) (EAT) (GOTO WORK) (GOTO GYM) (GOTO HOME) (EAT) (GOTO BED)
(WAKE-UP) (EAT) (GOTO GYM) (GOTO WORK) (GOTO HOME) (EAT) (GOTO BED) (SLEEP WALK)
Ready to start another cycle. The result of the last context was:

        CONTEXT 1
cogsav macrops
28.0   <M:1.01: WAKE-UP EAT (GOTO-GYM + ()) GOTO-WORK (GOTO-GYM + ()) GOTO-HOME EAT
       GOTO-BED>

The most interesting Partial macrops
14.0   <P: WAKE-UP EAT GOTO-GYM GOTO-WORK GOTO-HOME EAT GOTO-BED>
12.0   <P: WAKE-UP EAT GOTO-GYM GOTO-WORK GOTO-HOME EAT>
10.0   <P: WAKE-UP EAT GOTO-GYM GOTO-WORK GOTO-HOME>
8.0    <P: WAKE-UP EAT GOTO-GYM GOTO-WORK>
Observered trace of actions working with:
[M:1.01] (WAKE-UP) (GET-SNACK) (GOTO BED) [M:1.01] (SLEEP WALK)
All interesting macrops were discovered. This example is finished.
```

Figure 7.9: Student Example

optional. One does not need to workout in order to survive. Knowledge also indicates that a day must start by waking up in the morning, end by going to bed, and a student must also get some work done during the day, otherwise his/her advisor might become upset.

The system finds the required macrop for the student which is *wake up*, optionally *go to the gym*, *work*, optionally *go to the gym*, *head home*, *eat*, and *go to bed*. The noise of the student getting a midnight snack in the middle of the week and sleep walking at the end did not throw the system off track. Although this trace makes the problem look simple, it was not solvable without the domain knowledge[1]. Due to the amount of regularity in the example, a large number of macrops were created. The handling of these macrops slows the discovery process. The simple condition of

[1] Parameters were set to stop the system running without background knowledge after 500 agendas were executed. It was clear the system could run much longer before producing anything useful.

forcing days to start by waking up and end by going to bed is enough to prune the number of possible macrops so that the system can function. This example shows that the addition of simple knowledge can greatly improve the prospects of discovering the correct solution.

CHAPTER 8

FUTURE RESEARCH

As with most research projects, work on PLAND has introduced as many questions as it answered. In this chapter some of those open questions are discussed. The topics of the chapter include discovering fixed iteration loops, extending the fuzzy matching algorithm, processing overlapping macro operators, and incorporating more complex background knowledge into the system. The last topic will get special attention as other researchers are also working on combining similarity-difference based learning systems with explanation based learning systems.

8.1. Fixed Iteration Loops

The PLAND system overgeneralizes when it converts a series of juxtaposed strings into a loop macrop. There are two methods of avoiding this, besides not allowing any induction. First, more background knowledge could be used to control the generalization. Constraints of this type will be discussed in the last section of this chapter. Second, the amount of generalization could be reduced. One method of reducing the amount of generalization is to require more iterations of a sequence before constructing a loop macrop. Currently this is accomplished by setting a parameter of the loop discovery module. Another method of reducing over generalization is by allowing only the closing-the-interval generalization [Michalski83a] on the number of loop iterations. Thus, when the number of iterations for each loop occurrence is the same, the system simply replaces those iterations with a macrop of fixed size. The sequence is not converted into a Kleene closure loop where any number of iterations (greater than 1) is possible. When the system has observed a

different number of iterations of the same sequence, a range of the lowest to the highest number of iterations is allowed. All intervening occurrences are assumed to be acceptable under the generalization.

The system could handle the fixed iteration form of loop macrops with slight modifications to the internal representation. Making this change will greatly increase the time required to run data as the system will require more cpu cycles to perform subsumption checking. Recall that the subsumption of macrops is performed by using finite state machine representations of the macrops. A Kleene closure loop (the form of present loops) can be described in the same number of states as the length of the loop body. However, when a range of iterations is given, the number of states required to represent the machine is the product of the largest range delimiter and the length of the loop body. When one of the values of the range becomes large this will greatly increase the time required for the checking phase. Again more knowledge could be used to help determine when it is best to generalize the loop iterations.

8.2. Improved Matcher

Another area of research involves improving the fuzzy matching algorithm. A goal of such a matcher could be to require items within the pattern sequence to be the same. For instance, it would have been appropriate in the robot example to send the matcher a constraint that whenever some item is grasped it must be ungrasped as well and that no other grasp may take place until an ungrasp has occurred. Currently constraints of patterns apply only to a specific action. The type of pattern needed would have dependencies among a group of actions.

Determining how far one can push pattern matching generalization would be interesting. By taking small steps along the way the system is able to learn about the interconnections of the parts. Clearly knowledge to guide the generalizations is required. Two open questions are how much knowledge is required to perform pattern matching generalization in this manner and how accurate are the results produced by such a system. Intuitively it is appealing to have a system proceed in

such a fashion. The knowledge can guide the search by specifying allowable generalizations while the discoveries (or lack thereof) can be used to direct the knowledge accessed[1].

8.3. Overlapping Macrops

Overlapping macrops could be used to build new macro operators. In the simplest case this would require attaching two macrops together and removing the common portion. Removing the overlap can be difficult. When the overlapping portion of code is within a loop body or conditional, the process for merging is not straightforward. However, there are simple situations where it is beneficial to combine the macrops, such as merging *ABC* and *CDE* to get *ABCDE*. The new combined macrop could require much processing to be discovered from the primitives, but relatively easy to find using overlap. The current system keeps account of all interconnections between macrops in an ATMS-like structure. Contradictions indicate overlap and are maintained to facilitate the combining of the contradictory substructures.

While the above paragraph talks about actual overlap, there are cases when discovering a vague, nonexisting overlap is useful. If there is a macrop defined for *A B C* and another for *A D C* then the real macro operator might actually be *A (B + D) C*. The real macrop might not have been discovered due to the structure of the input example. Without some help from background knowledge it seems impossible to search through all the macrops discovered, determine how they could possibly fit together, and determine if the result is interesting. However, with appropriate domain knowledge such a search could be productive. Determining the type of knowledge required to perform this task is a research question.

8.4. More Background Knowledge

The major focus of future research on the PLAND system will be in the area of incorporating more knowledge into the system. More knowledge here means using knowledge-intensive

[1] Langley et al. discuss a similar notion and how it could be used in the BACON family [Langley87].

algorithms as in explanation based learning. More effective methods of combining explanation based and similarity-difference based approaches to learning will be explored. Other researchers are also exploring ways to effectively get these two types of learning to complement each other [Kodratoff87, Lebowitz85]. Kodratoff uses EBL techniques to drive the SDBL portion of the algorithm while Lebowitz uses SDBL techniques to guide the explanation building processes. It appears that this latter approach will more closely resemble the future directions of PLAND.

Lebowitz defines a single act of communication between the SDBL and the EBL components of his system. The proposed PLAND system will require many acts of communication between the cooperating modules. Each conversation should be on a different level in the generalization hierarchy or on a different solution path. The idea is that discovered macrops can be proved useful by an EBL type system. Also at this time some generalization might be done to the macrop, similar to that performed by the BAGGER system [Shavlik87a, Shavlik87b]. The EBL system would require domain knowledge on the effects of each action allowed in the environment (given example). The system would also require a high level notion of the task accomplished by the observed actions. It might additionally contain information on some methods useful in accomplishing the high level task. The goal of the system is to learn how the observed action accomplishes the task, along the way revealing previously unknown macrops. On the other hand, the system could learn a completely new method for performing the task from the observation. As the macrops are explained, the EBL system could guide the SDBL module's macrop search to confirm or deny its hypotheses.

Most complex tasks require many levels of generalization to explain them. PLAND currently works in these levels of generalization, and the proposed system should as well. The two components of the system would guide each other in such a scheme, the SDBL modules showing what actually exists in the example and the EBL modules proving generality of found items and suggesting generalizations to apply in order to confirm or deny possible goals. This area of research will involve controlling and defining the type of communications between these modules.

Another method of incorporating more knowledge into the current system is by integrating it with a planner. The planner could replace some complex background knowledge that might otherwise be required to verify found macrops. A planner knows what goals must be met to accomplish a task. Thus the planner can check the usefulness of discovered macrops, a function similar to that performed by a knowledge-intensive system but more sharply tuned to the problems involved in planning.

CHAPTER 9

CONCLUSIONS

The PLAND system discovers macro operators in plan sequences. The system uses domain independent algorithms for discovering loops, conditionals, and sequences in events that are connected by a single "follows" relation. This system shows that it is possible to learn macrops from an action trace, with or without background knowledge. PLAND is some of the first work done in the area of substructure discovery; undoubtedly more will follow.

Research on PLAND is concerned with two areas of research in artificial intelligence. The first is the area of substructure discovery. Building substructures allows the system to learn in at least two different ways. After the system discovers an interesting substructure, the substructure is learned at the knowledge level and can be used in building other substructures. A substructure also allows the system to assign properties to a portion of the event and thus can use those properties when performing conceptual clustering on the events. The property could be as simple as indicating that a node is in a certain type of substructure. This opens up the possibility for better clustering algorithms that will be able to get a more Gestalt fit to the data.

The second important area of research is combining knowledge-intensive algorithms, like explanation based learning, with similarity-difference based learning algorithms. It is clear that a truly intelligent system needs to use both in combination, but the most productive method for combining these methods is unknown. The PLAND system has laid the ground work for using knowledge-intensive algorithms with the less knowledge intensive similarity-difference based learning algorithms. Through the use of background knowledge the system can build contexts of generalization for the problem at hand. This allows the system to work at multiple levels in the

action hierarchy for a set of primitive actions. In this manner the system is able to abstract portions of the action sequence to reason about higher level goals. The goals dictate which actions are actually performed. In this scheme, the two algorithms work in conjunction. The SDBL portion discovers interesting components that actually exist in the example. The EBL portion generalizes those discovered substructures and gives direction for different areas to search and levels of abstraction to use. The system communicates repeatedly between the two modules to build the goal structure for the task.

Substructure discovery is interesting because of the small number of constraints. Recognizing structures is very important in allowing a system to reduce the complexity of common items and in thinking about old events in new ways. Discovering macrops can occur without the aid of specific background knowledge, but the macrops are pure syntactic entities. For the system to discover more complex and interesting macrops, background knowledge is required.

REFERENCES

[Andreae85] P. M. Andreae. "Justified Generalization: Aquiring Procedures from Examples." Technical Report 834 Ph.D. Thesis. MIT AI Lab. Cambridge. MA. January 1985.

[Bongard67] M. Bongard. *Pattern Recognition*. Spartan Books. New York. 1967.

[Cohen82] E. Cohen and E. A. Feigenbaum. *The Handbook of Artificial Intelligence, Volume III*. William Kaufman. Inc.. Los Altos. CA. 1982.

[DeJong86] G. F. DeJong and R. J. Mooney. "Explanation-Based Learning: An Alternative View." *Machine Learning 1*. 2 (April 1986). pp. 145-176. (Also appears as Technical Report UILU-ENG-86-2208. AI Research Group. Coordinated Science Laboratory. University of Illinois at Urbana-Champaign.)

[de Kleer86] J. de Kleer. "An Assumption-Based Truth Maintenance System." *Artificial Intelligence 28*. (1986). pp. 127-162.

[Dietterich83] T. G. Dietterich and R. S. Michalski. "A Comparative Review of Selected Methods for Learning from Examples." in *Machine Learning: An Artificial Intelligence Approach*. R. S. Michalski. J. G. Carbonell and T. M. Mitchell (ed.). Tioga Publishing Company. Palo Alto. CA. 1983. pp. 41-81.

[Dietterich86a] T. G. Dietterich and R. S. Michalski. "Learning to Predict Sequences." in *Machine Learning: An Artificial Intelligence Approach. Vol. II*. R. S. Michalski. J. G. Carbonell and T. M. Mitchell (ed.). Morgan Kaufmann. Los Altos. California. 1986. pp. 63-106.

[Dietterich86b] T. G. Dietterich. "Learning at the Knowledge Level." Technical Report. Oregon State University. Corvallis. OR. January 1986.

[Fikes72] R. E. Fikes. P. E. Hart and N. J. Nilsson. "Learning and Executing Generalized Robot Plans." *Artificial Intelligence 3*. (1972). pp. 251-288.

[Fisher85] D. Fisher. "A Proposed Method of Conceptual Clustering for Structured and Decomposable Objects." *Proceedings of the 1985 International Machine Learning Workshop*. Skytop. PA. June 1985. pp. 38-40.

[Gold67] M. Gold. "Language identification in the limit." *Information and Control 10*. (1967). pp. 447-474.

[Hayes-Roth78] F. Hayes-Roth and J. McDermott. "An interference matching technique for inducing abstractions." *Communications of the Association for Computing Machinery 21*. 5 (1978). pp. 401-410.

[Hoff83] W. A. Hoff. R. S. Michalski and R. E. Stepp. "INDUCE 3: A Program for Learning Structural Descriptions from Examples." Technical Report UIUCDCS-F-83-904. Department of Computer Science. University of Illinois. Urbana. IL. 1983.

[Holder87] L. B. Holder. "Discovering Substructures in Examples." M.S. Thesis (In Preparation). Department of Computer Science. University of Illinois. Urbana. IL. 1987.

[Hopcroft79] J. Hopcroft and J. Ullman. *Introduction to Automata Theory. Languages and Computation*. Addison-Wesley. Reading. MA. 1979.

[Kodratoff87] Y. Kodratoff and G. Tecuci. "What is an Explanation in Disciple?." *Proceedings of the Fourth International Workshop on Machine Learning*, Irvine,Calif., June 1987, pp. 160-166.

[Laird84] J. Laird, P. Rosenbloom and A. Newell. "Towards Chunking as a General Learning Mechanism." *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, August 1984, pp. 188-192.

[Langley81] P. Langley, G. L. Bradshaw and H. A. Simon. "BACON.5: The Discovery of Conservation Laws." *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, B.C., Canada, August 1981, pp. 121-126.

[Langley86] P. Langley, J. M. Zytkow, H. A. Simon and G. L. Bradshaw. "The Search for Regularity: Four Aspects of Scientific Discovery." in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, California, 1986, pp. 425-469.

[Langley87] P. Langley, H. A. Simon, G. L. Bradshaw and J. M. Zytkow. *Scientific Discovery: Computational Explorations of the Creative Processes*, MIT Press, Cambridge, MA, 1987.

[Lebowitz85] M. Lebowitz. "Integrated Learning: Controlling Explanation." Technical Report, Columbia University, New York, NY, July 1985.

[Lebowitz86] M. Lebowitz. "Concept Learning in a Rich Input Domain: Generalization-Based Memory." in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, California, 1986, pp. 193-214.

[Lenat84] D. B. Lenat and J. S. Brown. "Why AM and EURISKO Appear to Work." *Artificial Intelligence 23*, (1984), pp. 269-294.

[Michalski75] R. S. Michalski. "Variable-Valued Logic and its Applications to Pattern Recognition and Machine Learning." in *Multiple-Valued Logic and Computer Science*, D. Rine (ed.), North-Holland, New York, NY, 1975, pp. 506-534.

[Michalski83a] R. S. Michalski. "A Theory and Methodology of Inductive Learning." in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell, T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 83-134.

[Michalski83b] R. S. Michalski and R. E. Stepp. "Learning from Observation: Conceptual Clustering." in *Machine Learning: An Artificial Intelligence Approach*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Tioga Publishing Company, Palo Alto, CA, 1983, pp. 331-363.

[Michalski86] R. S. Michalski. "Understanding the Nature of Learning: Issues and Research Directions." in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufmann, Los Altos, California, 1986, pp. 3-25.

[Michalski87] R. S. Michalski, H. Ko and K. Chen. "Qualitative Prediction: A Method and Program SPARC/G." in *Expert Systems*, P. Dufour and A. Van Lamsweerde (ed.), Academic Press Inc. (forthcoming), London, 1987.

[Minton85] S. N. Minton. "Selectively Generalizing Plans for Problem-Solving." *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA, August 1985, pp. 596-599.

[Mitchell86] T. M. Mitchell, R. Keller and S. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning 1*, 1 (January 1986), pp. 47-80.

[Mogensen87] B. N. Mogensen, "Goal-Oriented Conceptual Clustering: The Classifying Attribute Approach," M.S. Thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, 1987.

[Pao78] T. Pao, J. C. III, M. Tuceryan and N. Ahuja, "Extracting Perceptual Structure in Dot Patterns: An Integrated Approach," *Computer Languages 3*, (January 1987), pp. 55-64, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign.

[Restle70] F. Restle, "Theory of Serial Pattern Learning: Structural Trees," *Psychological Review 77*, 6 (November 1970), pp. 481-495.

[Schank86] R. C. Schank, G. C. Collins and L. E. Hunter, "Transending inductive category formation in learning," *Behavioral and Brain Sciences 9*, (1986), pp. 639-686.

[Schuegraf74] E. J. Schuegraf and H. S. Heaps, "A comparison of algorithms for data base compression by use of fragments as language elements," *Information Storage and Retrieval 10*, (1974), pp. 309-319.

[Shavlik87a] J. W. Shavlik and G. F. DeJong, "An Explanation-Based Approach to Generalizing Number," *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, Milan, Italy, August 1987.

[Shavlik87b] J. W. Shavlik and G. F. DeJong, "BAGGER: An EBL System that Extends and Generalizes Explanations," *Proceedings of the National Conference on Artificial Intelligence*, Seattle, WA, July 1987.

[Simon63] H. A. Simon and K. Kotovsky, "Human Acquistion of Concepts for Sequential Patterns," *Psychological Review 70*, 6 (1963), pp. 534-546.

[Stepp83] R. E. Stepp, "A Description and User's Guide for CLUSTER/2, A Program for Conjunctive Conceptual Clustering," Report No. UIUCDCS-R-83-1084., Department of Computer Science, University of Illinois, Urbana, IL, November 1983.

[Stepp84] R. E. Stepp, "Conjunctive Conceptual Clustering: A Methodology and Experimentation," Ph.D. Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, 1984.

[Stepp86] R. E. Stepp and R. S. Michalski, "Conceptual Clustering: Inventing Goal-Oriented Classifications of Structured Objects," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell and T. M. Mitchell (ed.), Morgan Kaufman, 1986, pp. 471-498.

[Treisman82] A. Treisman, "Perceptual Grouping and Attention in Visual Search for Features and for Objects," *Journal of Experimental Psychology: Human Perception and Performance 8*, 2 (1982), pp. 194-214.

[Tversky77] A. Tversky, "Features of Similarity," *Psychological Review 84*, 4 (July 1977), pp. 327-352.

[Vere78] S. A. Vere, "Inductive Learning of Relational Productions," in *Pattern Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (ed.), Academic Press, New York, 1978.

[Wattenmaker87] W. D. Wattenmaker, G. L. Nakamura and D. L. Medin, "Relationships Between Similarity-based and Explanation-based Categorization," in *Contemporary Science and Natural Explanations: Common Sense Concepts of Causality*, D. Hilton (ed.), Harvester Press, Sussex, England, 1987, pp. 205-241.

[Winston75] P. H. Winston, "Learning Structural Descriptions from Examples," in *The Psychology of Computer Vision*, P. H. Winston (ed.), McGraw-Hill, New York, NY, 1975, pp. 157-210.

[Winston84] P. H. Winston, *Artificial Intelligence (Second Edition)*, Addison-Wesley, Reading, MA, 1984.

[Wolff76] J. G. Wolff, "Frequency, Conceptual Structure and Pattern Recognition," *British Journal of Psychology 67*, 3 (1976), pp. 377-390.

[Wolff82] J. G. Wolff, "Language Acquistion, Data Compression and Generalization," *Language and Communication 2*, 1 (1982), pp. 57-89.

[Zahn71] C. T. Zahn, "Graph-Theoretical Methods for Detecting and Describing Gestalt Clusters," *IEEE Transactions on Computers C-20*, 1 (January 1971), pp. 68-86.

END

DATE

FILMED

FEB.

1988